

高职高专计算机规划教材

# ASP.NET 3.5 教程

郑阿奇 主 编

電子工業出版社

**Publishing House of Electronics Industry**

北京 • BEIJING

## 内 容 简 介

ASP.NET 3.5 是最新可视化 Web 开发平台, 本书结合 ASP.NET 教学和应用开发的经验, 系统介绍了 ASP.NET 3.5 及其应用开发。本书内容包括教程、配套的实验和综合应用实习三部分, 其中教程包括 11 章, 比较系统地介绍了 ASP.NET 3.5, 包括 ASP.NET 开发技术概述、C# 语言基础、C# 面向对象编程、ASP.NET 应用程序基础和内置对象、ASP.NET 服务器控件和客户端脚本、网站设计、ASP.NET 数据库编程、文件 I/O 与流处理、ASP.NET 高级技术、Web 服务和 ASP.NET AJAX 等。教程中实例的选择考虑在一定的应用性的前提下, 采用较小的规模。实验部分也同时考虑能够方便理解, 最后的综合应用实习旨在训练解决问题。

本书专为高职高专进行设计, 可作为高职高专有关课程教材, 也可作为广大学习 ASP.NET 语言的人员参考。本套教程可免费下载教学课件、教程、实验和综合应用源程序。

未经许可, 不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有, 侵权必究。

### 图书在版编目(CIP)数据

ASP.NET 3.5 教程 / 郑阿奇主编. —北京: 电子工业出版社, 2009.10

高职高专计算机规划教材

ISBN 978-7-121-09503-0

I. A… II. 郑… III. 主页制作—程序设计—高等学校: 技术学校—教材 IV. TP393.092

中国版本图书馆 CIP 数据核字 (2009) 第 160902 号

策划编辑: 赵云峰

责任编辑: 韩玲玲

装 订:

印 刷:

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 787×1092 1/16 印张: 21.5 字数: 544 千字

印 次: 2009 年 10 月第 1 次印刷

印 数: 4 000 册 定价: 29.00 元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888。

质量投诉请发邮件至 [zlts@phei.com.cn](mailto:zlts@phei.com.cn), 盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线: (010) 88258888。

# 前 言

微软公司在 2000 年推出了 .NET 战略，它是微软面向互联网时代构筑的新一代平台，是微软在 21 世纪初的一个重大战略步骤。2002 年发布了 .NET Framework 1.0 正式版，2005 年发布了 .NET Framework 2.0 正式版，2008 年发布了 .NET Framework 3.5 正式版，Microsoft Visual Studio 2008 是其最新的可视化开发平台，ASP.NET 3.5 是其中最新的可视化 Web 开发平台。本书结合 ASP.NET 教学和应用开发的经验，系统介绍了 ASP.NET 3.5 及其应用开发。

本书包括教程、配套的实验和实习三部分。教程包括 11 章，比较系统地介绍了 ASP.NET 3.5，包括 ASP.NET 开发技术概述、C# 语言基础、C# 面向对象编程、ASP.NET 应用程序基础与内置对象、ASP.NET 服务器控件和客户端脚本、网站设计、ASP.NET 数据库编程、文件 I/O 与流处理、ASP.NET 高级技术、Web 服务和 ASP.NET AJAX 等。

ASP.NET 3.5 内容多，本书尽可能介绍其各个方面的主要内容，并且通俗易懂。实例的选择考虑在一定的应用性的前提下，采用较小的规模。实验也同时考虑能够方便理解，最后的实习部分就是训练解决问题。

本书吸取了以前编写 ASP.NET 教程的经验。只要阅读本书，结合实验进行练习，就能在较短的时间内基本掌握 ASP.NET 3.5 及其应用技术，欢迎读者比较选择。

本书同步配套 PowerPoint 课件、书中的源代码和比较完整的应用系统，需要者可从 [http:// www.hxedu.com.cn](http://www.hxedu.com.cn) 网站上免费下载。

本书由南京师范大学郑阿奇主编，其他几个同志参加了本书的基础工作。在此一并表示感谢！

由于作者水平有限，书中错误在所难免，欢迎广大读者批评指正！

作者 E-mail: [easybooks@163.com](mailto:easybooks@163.com).

编 著 者

# 目 录

## 第一部分 教程

第 1 章 ASP.NET 开发技术概述 .....	(1)
1.1 Web 工作原理 .....	(1)
1.1.1 Web 服务器 .....	(1)
1.1.2 Web 客户端 .....	(2)
1.2 HTML 与 HTTP .....	(2)
1.2.1 HTML 标记语言 .....	(2)
1.2.2 XHTML 标记语言 .....	(4)
1.2.3 CSS 样式表 .....	(7)
1.2.4 脚本语言 .....	(11)
1.2.5 HTTP 协议与 URL .....	(12)
1.2.6 HTML 表单与 Web 请求 .....	(15)
1.3 Web 开发技术的发展 .....	(17)
1.3.1 静态网页与动态网页 .....	(17)
1.3.2 客户端动态技术 .....	(17)
1.3.3 服务器端动态技术 .....	(18)
1.4 .NET Framework 与 ASP.NET .....	(19)
1.4.1 .NET Framework .....	(19)
1.4.2 ASP.NET .....	(20)
1.5 ASP.NET 应用程序的组成 .....	(21)
1.5.1 虚拟目录 .....	(21)
1.5.2 网页文件 .....	(22)
1.5.3 网站配置文件 (Web.config 文件) .....	(22)
1.5.4 网站全局文件 (Global.asax 文件) .....	(22)
1.5.5 保留文件夹 .....	(23)
1.6 VS 2008 集成开发工具简介 .....	(23)
1.6.1 编译和运行 Web 应用程序 .....	(23)
1.6.2 部署 Web 应用程序 .....	(24)
1.6.3 使用帮助系统 .....	(25)
1.7 简单的 ASP.NET 程序实例 .....	(26)
1.7.1 创建 Web 应用程序的一般步骤 .....	(26)
1.7.2 一个简单的 ASP.NET 程序实例 .....	(26)
习题 .....	(29)
第 2 章 C#语言基础 .....	(31)
2.1 C#语言概述 .....	(31)
2.2 数据类型 .....	(31)
2.2.1 值类型 .....	(31)

2.2.2	引用类型 .....	(34)
2.2.3	装箱和拆箱 .....	(36)
2.3	常量和变量 .....	(37)
2.3.1	常量 .....	(37)
2.3.2	变量 .....	(40)
2.4	运算符和表达式 .....	(41)
2.4.1	算术运算符 .....	(41)
2.4.2	关系运算符 .....	(42)
2.4.3	逻辑运算符 .....	(43)
2.4.4	赋值运算符 .....	(44)
2.4.5	条件运算符 .....	(45)
2.4.6	运算符的优先级与结合性 .....	(46)
2.5	流程控制 .....	(47)
2.5.1	条件语句 .....	(47)
2.5.2	循环语句 .....	(51)
2.5.3	跳转语句 .....	(55)
2.5.4	异常处理 .....	(59)
2.6	数组、结构和枚举 .....	(60)
2.6.1	数组 .....	(60)
2.6.2	结构 .....	(65)
2.6.3	枚举 .....	(66)
	习题 .....	(68)
第3章	C#面向对象编程 .....	(71)
3.1	类和对象 .....	(71)
3.1.1	创建类和对象 .....	(71)
3.1.2	构造函数和析构函数 .....	(74)
3.2	方法 .....	(78)
3.2.1	方法的声明 .....	(78)
3.2.2	方法的参数 .....	(79)
3.2.3	静态方法与实例方法 .....	(81)
3.2.4	方法的重载 .....	(83)
3.3	属性 .....	(84)
3.4	继承和多态 .....	(86)
3.4.1	继承 .....	(87)
3.4.2	多态 .....	(90)
3.5	委托和事件 .....	(93)
3.5.1	委托 .....	(93)
3.5.2	事件 .....	(94)
3.6	接口 .....	(96)
3.7	集合 .....	(97)

3.7.1	使用 Array 类进行排序与查找.....	(98)
3.7.2	使用 Stack 类.....	(99)
3.8	命名空间和局部类.....	(101)
3.8.1	命名空间.....	(101)
3.8.2	局部类.....	(102)
	习题.....	(104)
第 4 章	ASP.NET 应用程序基础与内置对象.....	(105)
4.1	ASP.NET 应用程序基础.....	(105)
4.1.1	aspx 代码模式和页面元素.....	(105)
4.1.2	页面指令.....	(106)
4.1.3	页生命周期.....	(109)
4.2	ASP.NET 内置对象.....	(110)
4.2.1	Response 对象.....	(110)
4.2.2	Request 对象.....	(112)
4.2.3	Server 对象.....	(117)
4.2.4	Application 对象.....	(119)
4.2.5	Session 对象.....	(121)
4.2.6	Page 对象.....	(124)
4.2.7	Cache 对象.....	(126)
	习题.....	(127)
第 5 章	ASP.NET 服务器控件和客户端脚本.....	(128)
5.1	控件概述.....	(128)
5.2	HTML 服务器控件.....	(129)
5.2.1	HTML 服务器控件的层次结构.....	(129)
5.2.2	HTML 服务器控件的基本语法.....	(129)
5.2.3	HTML 服务器控件的应用.....	(130)
5.3	Web 服务器控件.....	(133)
5.3.1	Web 服务器控件的层次结构.....	(133)
5.3.2	Web 服务器控件的基本语法.....	(134)
5.3.3	Web 服务器控件的属性.....	(135)
5.3.4	Web 服务器控件的事件.....	(136)
5.3.5	标准控件详解.....	(136)
5.4	验证控件.....	(153)
5.4.1	客户端验证和服务端验证.....	(153)
5.4.2	验证控件分类及作用.....	(153)
5.4.3	验证控件详解.....	(154)
5.4.4	关闭客户端验证功能.....	(158)
5.5	用户控件.....	(158)
5.5.1	建立用户控件.....	(158)
5.5.2	使用用户控件.....	(159)

习题 .....	(160)
<b>第6章 网站设计</b> .....	(161)
6.1 母版页和内容页 .....	(161)
6.1.1 母版页和内容页概述 .....	(161)
6.1.2 创建母版页和内容页 .....	(163)
6.1.3 访问母版页控件和属性 .....	(165)
6.2 主题和皮肤 .....	(166)
6.2.1 主题概述 .....	(166)
6.2.2 创建主题 .....	(169)
6.2.3 应用主题 .....	(172)
6.3 网站导航 .....	(174)
6.3.1 站点地图和 SiteMapPath 控件 .....	(174)
6.3.2 用 Menu 控件导航 .....	(176)
6.3.3 用 TreeView 控件导航 .....	(177)
习题 .....	(178)
<b>第7章 ASP.NET 数据库编程</b> .....	(179)
7.1 数据库基础 .....	(179)
7.1.1 数据库和数据库管理系统 .....	(179)
7.1.2 表和视图 .....	(180)
7.1.3 用 VS 2008 创建数据库和表 .....	(181)
7.1.4 SQL 语言 .....	(182)
7.2 数据访问技术 .....	(186)
7.2.1 数据访问概述 .....	(186)
7.2.2 数据源控件简介 .....	(186)
7.2.3 数据绑定控件简介 .....	(187)
7.3 数据源控件 .....	(188)
7.3.1 SqlDataSource 控件 .....	(188)
7.3.2 AccessDataSource 控件 .....	(191)
7.3.3 XmlDataSource 控件 .....	(191)
7.3.4 SiteMapDataSource 控件 .....	(192)
7.3.5 ObjectDataSource 控件 .....	(192)
7.3.6 LinqDataSource 控件 .....	(195)
7.4 数据绑定控件 .....	(196)
7.4.1 GridView 控件 .....	(196)
7.4.2 DetailsView 控件 .....	(197)
7.4.3 FormView 控件 .....	(199)
7.4.4 ListView 控件 .....	(200)
7.4.5 内部数据绑定语法 .....	(201)
7.5 ADO.NET 数据访问编程模型 .....	(203)
7.5.1 ADO.NET 数据访问模型简介 .....	(203)

7.5.2 数据集 .....	(204)
7.5.3 数据提供程序 .....	(208)
习题 .....	(213)
第 8 章 文件 I/O 与流处理 .....	(215)
8.1 文件系统操作 .....	(215)
8.1.1 使用驱动器 .....	(215)
8.1.2 文件夹操作 .....	(217)
8.1.3 文件操作 .....	(220)
8.1.4 使用路径 .....	(223)
8.2 文件读写操作 .....	(224)
8.2.1 使用 FileStream 类读写文件 .....	(225)
8.2.2 使用 Reader/Writer 类读写文件 .....	(226)
8.2.3 文件压缩 .....	(228)
8.3 文件上传 .....	(230)
习题 .....	(231)
第 9 章 ASP.NET 高级技术 .....	(232)
9.1 ASP.NET 配置 .....	(232)
9.1.1 ASP.NET 配置概述 .....	(232)
9.1.2 配置文件的结构 .....	(232)
9.1.3 常用配置 .....	(233)
9.2 高速缓存 .....	(236)
9.2.1 ASP.NET 缓存概述 .....	(236)
9.2.2 页面输出缓存 .....	(238)
9.2.3 页面部分缓存 .....	(239)
9.2.4 应用程序数据缓存 .....	(240)
9.3 ASP.NET XML 编程 .....	(243)
9.3.1 XML 基本概念 .....	(243)
9.3.2 XML 数据访问 .....	(245)
习题 .....	(249)
第 10 章 Web 服务 .....	(250)
10.1 Web 服务的基本概念 .....	(250)
10.1.1 基于组件的分布式计算概念 .....	(250)
10.1.2 什么是 Web 服务 .....	(250)
10.2 ASP.NET Web 服务的创建与测试 .....	(251)
10.2.1 创建 Web 服务 .....	(251)
10.2.2 @ WebService 指令 .....	(252)
10.2.3 Web 服务类 .....	(252)
10.2.4 WebService 特性 .....	(253)
10.2.5 定义 Web 服务方法 .....	(253)
10.2.6 测试 Web 服务 .....	(253)



10.3	使用 ASP.NET Web 服务 .....	(254)
10.3.1	添加 Web 引用 .....	(254)
10.3.2	客户端调用 Web 服务 .....	(255)
习题	.....	(256)
第 11 章	ASP.NET AJAX .....	(257)
11.1	ASP.NET AJAX 概述 .....	(257)
11.1.1	为什么使用 AJAX .....	(257)
11.1.2	VS 2008 与 ASP.NET AJAX .....	(258)
11.1.3	ASP.NET AJAX 客户端技术 .....	(258)
11.1.4	ASP.NET AJAX 服务器端技术 .....	(258)
11.2	建立 ASP.NET AJAX 应用程序 .....	(258)
11.3	ASP.NET AJAX 服务器端控件 .....	(259)
11.3.1	ScriptManager 控件 .....	(260)
11.3.2	ScriptManagerProxy 控件 .....	(260)
11.3.3	UpdatePanel 控件 .....	(260)
11.3.4	Timer 控件 .....	(262)
11.3.5	UpdateProgress 控件 .....	(263)
习题	.....	(264)
第二部分	实验	
实验 1	创建与发布 ASP.NET 应用程序 .....	(265)
实验 2	C#语言基础应用 .....	(268)
实验 3	面向对象编程 .....	(272)
实验 4	内置对象的应用 .....	(276)
实验 5	ASP.NET 服务器控件应用 .....	(280)
实验 6	母版、主题和导航设计 .....	(284)
实验 7	数据库编程 .....	(288)
实验 8	文件系统访问 .....	(293)
实验 9	ASP.NET XML 编程 .....	(297)
实验 10	Web 服务设计 .....	(302)
实验 11	AJAX 应用 .....	(307)
第三部分	实习	
综合应用实例：BBS 系统 .....		(309)
P.1	系统功能设计 .....	(309)
P.2	系统流程设计 .....	(309)
P.3	数据库设计 .....	(310)
P.4	母版页设计 .....	(312)
P.4.1	添加站点地图 .....	(312)
P.4.2	添加母版页 .....	(313)
P.5	主题设计 .....	(313)
P.6	全局变量 .....	(314)
P.7	注册模块设计 .....	(315)

P.7.1 服务条款页面设计..... (315)

P.7.2 用户注册页面设计..... (315)

P.8 登录模块设计 ..... (317)

    P.8.1 用户登录页面设计..... (317)

    P.8.2 管理员登录页面设计..... (318)

P.9 发帖模块设计 ..... (319)

    P.9.1 查询主帖页面设计..... (320)

    P.9.2 发表新帖页面设计..... (321)

    P.9.3 查看详细信息页面设计..... (323)

P.10 回帖模块设计..... (326)

P.11 管理帖模块设计..... (327)

P.12 系统扩展 ..... (330)

# 第一部分 教程

## 第 1 章 ASP.NET开发技术概述

微软公司在 2000 年推出了 .NET 战略，它是微软面向互联网时代构筑的新一代平台，是微软在 21 世纪初的一个重大战略步骤；2002 年发布了 .NET Framework（.NET 框架）1.0 正式版，2005 年发布了 .NET Framework 2.0 正式版，2008 年发布了 .NET Framework 3.5 正式版。随着 .NET Framework 2.0 的发布，.NET 技术开始走向成熟，尤其是用于 Web 应用程序开发的核心技术。最新版的 ASP.NET 3.5 更是备受关注。本章将介绍 Web 的基本工作原理和 Web 通信的本质，同时还将介绍 ASP.NET 3.5 和 VS 2008 开发环境的相关知识，使读者对网页编程技术有基本的认识，为今后编写 Web 应用程序打下基础。

### 1.1 Web工作原理

WWW（World Wide Web，万维网）由遍布在互联网中被称为 Web 服务器的计算机和安装了 Web 浏览器软件的计算机组成，它是一种基于超文本方式工作的信息系统。作为一个能够处理文字、图像、声音、视频等多媒体信息的综合系统，它提供了丰富的信息资源，这些信息资源以 Web 页面的形式，分别存放在各个 Web 服务器上，用户可以通过浏览器选择并浏览所需的信息。

#### 1.1.1 Web服务器

所谓 Web 服务器，即安装了 Web 服务器软件的计算机。Web 服务器软件对外提供 Web 服务，供客户访问浏览。实际上，Web 服务器软件的本质与其他各种提供网络服务的软件一样，接收客户端请求，然后将特定的内容返回客户端。

Web 服务器的工作流程是：用户通过 Web 浏览器向 Web 服务器请求一个资源，当 Web 服务器接收到这个请求后，将替用户查找该资源，然后将结果返回给 Web 浏览器。所请求的资源的内容多种多样，可以是普通的 HTML 页面、音频文件、视频文件或图片等。Web 服务器的工作流程如图 1.1 所示。

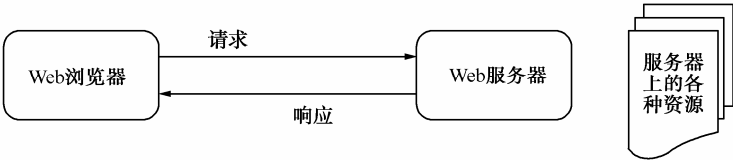


图 1.1 Web 服务器的工作流程

1.1.2 Web客户端

Web 客户端是一个比较宽泛的概念，通常将那些向 Web 服务器发送请求以获取资源的软件称为 Web 客户端。Web 客户端可以是各种类型的软件，目前被广泛使用的便是 Web 浏览器，如微软公司的 IE 浏览器、firefox（火狐）、Maxthon（傲游）等。

Web 客户端的功能是：根据客户的请求，发送特定的资源请求给 Web 服务器，当接收到 Web 服务器的响应后，将响应的内容按预先定义的形式显示出来。

首先，用户单击超链接或在浏览器地址栏中输入网页的地址，此时浏览器将该信息转换成标准的 HTTP 请求并发送给 Web 服务器。其次，当 Web 服务器接收到 HTTP 请求后，根据请求的内容，查找所需的信息资源，找到相应的资源后，Web 服务器将该部分资源通过标准的 HTTP 响应发送回浏览器。最后，浏览器接收到响应后，将 HTML 文档显示出来。一个基本的请求过程如图 1.2 所示。

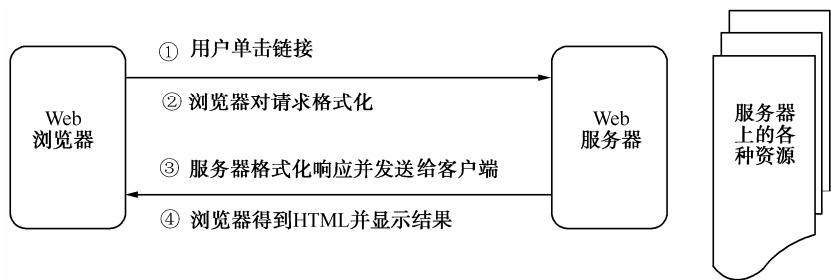


图 1.2 基本的请求过程

1.2 HTML与HTTP

前面介绍了 Web 访问过程中的两个重要的组成部分和各自的功能，下面介绍 Web 客户端和 Web 服务器之间传输的内容和协议。

1.2.1 HTML标记语言

HTML（HyperText Markup Language）是一种简单、通用的超文本标记语言，Web 服务器返回给客户端的最常见的内容就是 HTML 文档。可以用其制作包含图像、文字、声音等精彩内容在内的网页，它由浏览器解释并显示在用户浏览器的窗口中。HTML 文档的基本结构如下所示：

```
<html>
<head> ...</head>           //文档头部分
<body >
...                           //文档的主体部分
</body >
</html>
```

基本 HTML 页面从<html>标记开始，到</html>结束。在它们之间，就是头部分和主体部分。头部分用<head>...</head>标记（Tags）界定，一般包含网页标题、文档属性等不在页

面上显示的元素。主体部分是网页的主体，内容均会反映在页面上，用<body>....</body>标记来界定，主要包括文字、图像、动画、超链接等内容。

注意：代码中“//”符号为注释符。

标记是 HTML 语言的标签符号和用标签符号构成的各种元素的统称。标签是描述性的标记，用一对“<>”中间包含若干字符表示，通常成对出现，前一个是起始标签，后一个为结束标签。标记不区分大小写。较常见的标签如下：

<html></ html >	//HTML 文档的开始和结束标签
<title></ title >	//HTML 文档标题的开始和结束标签
<body></ body >	//HTML 文档体的开始和结束标签
<table></ table >	//表格的开始和结束标签
<tr></ tr >	//表格中行的开始和结束标签
<td></ td >	//表格中列的开始和结束标签
<a></ a >	//超链接的开始和结束标签
<p></ p >	//段落的开始和结束标签
<form></form>	//为用户输入创建表单标签
 	//换行标签强行中断当前行，使后续内容在下一行显示
<input></input>	//定义输入域的开始，在其中用户可输入数据

【例 1-1】 用 HTML 语言设计一个简单的登录网页页面，设计后打开页面如图 1.3 所示。



图 1.3 简单的 HTML 示例

设计步骤如下：

(1) 新建一个命名为“example1-1”的文本文件，把下面的代码添加到 example1-1.txt 文件中并保存。

```
<HTML>
<HEAD>
<Title>简单的 HTML 示例</Title>
</HEAD>
<BODY >
<form name="form1" method="post" id="form1">
    <p align="center">用户登录</p><br>
    <table align="center" border="1" width="40%">
        <tr>
            <td align="right">用户名: </td>
            <td><input name="TextBoxName" type="text" id="TextBoxName" /></td>
```

```

        </tr>
      <tr>
        <td align="right">密码: </td>
        <td><input name="TextBoxPwd" type="text" id="TextBoxPwd" /></td>
      </tr>
    </table>
    <p align="center"><input type="submit" name="BtnOK" value="确定" id="BtnOK" /></p>
  </form>
</BODY>
</HTML>

```

其中标签中的“name”、“method”、“id”等为标签的属性，等号后的为属性值。

在 HTML 的所有标记中，许多标记有若干属性，通过设置属性值，可对标记内的内容进行控制，多个属性之间必须用空格隔开。如果不设置标记的属性值，则使用系统的默认属性值。例如，上例中的 body、p 标记。

属性设置的一般格式为

```
<标记 属性 1="值 1" 属性 2="值 2"...> ... </标记>
```

(2) 更改 example1-1.txt 的后缀名，改成 example1-1.html。用浏览器打开它，浏览器会对文档中的不同标记进行解释并显示指定的效果，打开后的网页页面即如图 1.3 所示。

注意：有些标记没有结束标记，如代码中的<br>标记。

## 1.2.2 XHTML 标记语言

HTML 从出现到现在，标准不断完善，功能也越来越强大，但其规范化要求不是很严格，仍有很多缺陷和不足。例如，代码琐碎、臃肿，尤其是标记使用不规范，浏览器需要有足够的能力才能够正确显示 HTML 页面。随着 Web 的发展，当面对多样的数据和表现，如数学公式、音乐标注等非传统文档类型时，HTML 显得力不能及。另外，HTML 不能适应越来越多的网络设备和应用的需要，如手机、PDA 等显示设备不能直接显示 HTML。因此 HTML 的后继者 XHTML 就出现了。XHTML 非常严格，兼容性强，交互性好，能够解决 HTML 发展的问题，如多种显示设备的支持、多样数据的表现等。

### 1. XHTML 的格式

XHTML 的格式与 HTML 类同，只是文档前面加了一个文档说明和标记的命名空间。它的基本格式为

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> //XHTML 文档类型的声明
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>XHTML 的格式</title> //文档头部分
</head>
<body>
    <div>XHTML 文档的主体部分</div> //文档主体部分

```

```
</body>
</html>
```

第一行代码是 DOCUMENT 声明。DOCUMENT 是 document type（文档类型）的缩写，用来说明所使用的 XHTML 是什么版本。上例代码中所使用的是 xhtml1-transitional 文档类型。文档类型中包含了文档的规则，浏览器展现页面时会根据页面所采用的 DTD（文档类型定义）来解释页面内的标识，并将其显示出来。

第二行代码与 HTML 也有明显差别。在 HTML 代码中仅有<html>，但在 XHTML 中，标记内还有 xmlns="http://www.w3.org/1999/xhtml"。"xmlns"是 XHTML namespace 的缩写，即 XHTML 命名空间，它用来声明网页内所用到的标记是属于哪个命名空间的。在不同的命名空间中，可以有相同的标记表示不同的含义，所以声明命名空间是非常必要的。

2. XHTML与HTML的区别

(1) 标记的嵌套使用

标记的嵌套使用是对文档结构的要求。尽管 HTML 也要求正确的嵌套，可是在实际中，即使 HTML 使用了不正确的嵌套形式，很多浏览器也一样可以正常显示。但是 XHTML 对文档的结构要求较严格，整个文档一定要有正确的组织格式，所有的嵌套必须完全正确。例如，在 XHTML 中：

```
<body><div>XHTML 文档的主体部分</div> </body>      //正确的嵌套
<body><div>XHTML 文档的主体部分</body></div>      //错误的嵌套
```

(2) 大小写的使用

HTML 是不区分大小写的，元素和属性名称可以是大写、小写或是混合书写。但是 XHTML 文档要求所有的元素和属性名称必须要小写，属性值不做要求。例如，在 XHTML 中：

```
<img src = "/images/025.jpg" alt = "nanjing" />      //正确
<IMG SRC = "/images/025.jpg" Alt = "nanjing" />      //错误
```

(3) 引号的使用

HTML 中的引号比较随意，属性值可以用引号引起来，也可以不使用引号。但是 XHTML 中要求所有的属性值都必须加引号。例如，在 XHTML 中：

```
<input type="submit" value="确定" id="BtnOK" />      //正确
<input type= submit value=确定 id=BtnOK"/>          //错误
```

(4) 结束标记

在 HTML 中，有些标记是可以省略结束标记的，由下一个标记的出现来让它自然结束。这种省略形式在 XHTML 中是绝对不允许的，XHTML 要求所有的标记都必须有结束标记，即使是单个标记，也需要使用“/>”来结束。例如，在 XHTML 语言中：

```
<p> asp.net3.5      //错误
<p>asp.net3.5</p>   //正确
asp.net3.5<br>      //错误
asp.net3.5<br/>     //正确
```

(5) 样式的使用

在不使用样式表的情况下，HTML 中的每个属性都可以直接使用“属性名=属性值”的方法设置外观样式。例如，在 HTML 中：

```

```

上例中的图片的高度和宽度样式是直接通过属性设置的。但在 XHTML 中，如果不使用样式表，则只能通过 style 属性来设置样式。上述代码在 XHTML 中如下所示：

```

```

(6) id 和 name

在 HTML 中，每个元素都可以定义 name 属性，同时也可以定义 id 属性，两个属性都可以标识某一个元素。例如：

```
<input type="submit" value="确定" name="BtnOK" id="BtnOK" />
```

但是在 XHTML 中，每个元素只能有一个标记属性 id，name 属性不再被使用。例如：

```
<input type="submit" value="确定" id="BtnOK" /> //没有 name 属性
```

(7) 特殊符号

在 XHTML 中，特殊符号用编码表示，例如：

```
&lt;    表示 <
&gt;   表示 >
&amp; 表示 &
```

**【例 1-2】** 用 XHTML 语言改写【例 1-1】，用于体现 XHTML 与 HTML 的区别。  
设计步骤如下：

(1) 新建一个命名为“example1-2”的文本文件，把下面的代码添加到 example1-2.txt 文件中并保存。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>简单的 HTML 示例</title>
</head>
<body>
<form name="form1" method="post" id="form1">
    <p align="center">用户登录</p><br>
    <table align="center" border="1" width="40%">
        <tr>
            <td align="right">用户名: </td>
            <td><input name="TextBoxName" type="text" id="TextBoxName" /></td>
        </tr>
        <tr>
            <td align="right">密码: </td>
            <td><input name="TextBoxPwd" type="text" id="TextBoxPwd" /></td>
```



```
        </tr>
    </table>
    <p align="center"><input type="submit" value="确定" id="BtnOK" /></p>
</form>
</body>
</html>
```

(2) 更改 example1-2.txt 的后缀名, 改成 example1-2.html。用浏览器打开它, 浏览器会对文档中的不同标记进行解释并显示指定的效果, 打开后的网页页面即如图 1.3 所示。

比较【例 1-1】和【例 1-2】可以看出, XHTML 语言多了 DOCUMENT 声明和 XHTML 命名空间, 同时文档中的元素和属性名称小写, 并去除了 name="BtnOK"。

### 1.2.3 CSS样式表

在网页中另一个重要的表现技术就是 CSS (Cascading Style Sheet, 层叠样式表)。它是 W3C (World Wide Web Consortium, 全球万维网联盟) 为弥补 HTML 在显示属性设定上的不足而制定的一套扩展样式标准。CSS 标准中重新定义了 HTML 中原来的文字显示样式, 增加了类、层等新概念, 可以对文字重叠、定位等提供更为丰富的样式。在当今的网页制作中, 几乎所有漂亮的网页都使用了 CSS。有了 CSS 的控制, 网页便会给人一种赏心悦目的感觉。

#### 1. CSS样式规则

样式是指元素在浏览器中的呈现形式, 如元素的高度、宽度, 是否有边框, 边框颜色, 边框粗细, 字体大小, 字体颜色, 元素的背景色和背景图片, 元素内数据的对齐方式等。在 (HTML 或 XHTML) 标记中可以通过 style 属性设置元素的样式, 每个 style 内包含一个或多个属性, 其一般形式为:

```
style="属性名 1: 属性值 1; 属性名 2: 属性值 2; ....."
```

属性名与属性值之间用冒号 “:” 分隔, 如果一个样式有多个属性, 则各属性之间用分号 “;” 隔开。例如:

```
<p style="font-family: '宋体'; color:green; background-color: yellow; font-size: 9pt">.....</p>
```

这种通过元素的 style 属性给元素设置样式的优点是方便、直观, 缺点是需要给每个元素设置。为了解决此问题, 可以进行集中样式管理, 这种单独定义的样式被称为样式规则。

样式必须符合的格式如下所示:

```
样式选择符{ 样式属性 1:值 1; 样式属性 2:值 2; 样式属性 3:值 3; .....
```

其中包含两个部分: 样式选择符和样式属性。样式选择符用来设置样式所作用的元素范围。{ } 内是所采用的样式。{ } 内的样式会对样式选择符作用范围内的所有元素生效。例如:

```
td {font-size: 12px; background-color:green}
```

td 表示样式选择符, 对<td>标签起作用。font-size 和 background-color 表示样式属性, 分别表示字体大小和背景色。12px 和 green 则表示属性值, 字体为 12 像素, 背景色为绿色。

## 2. 样式选择符分类

为了方便控制样式规则所作用的元素范围，CSS 中把样式选择符分为如下几类：标签选择符、类选择符、ID 选择符、伪类选择符、包含选择符和并列选择符。

### (1) 标签选择符

标签选择符（即 HTML 或 XHTML 标记）是最典型的选择符类型。定义的时候直接使用标记名称作为选择符名称。常见格式如下：

```
标签名{ 样式属性 1:值 1;  样式属性 2:值 2; 样式属性 3:值 3; ..... }
```

例如：

```
div
{
    background-color: white;      //背景颜色
    text-align: center;          //文本水平布局
}
```

引用本 css 文件的页面内所有 div 标记都会采用{}内的样式。

### (2) 类选择符

标签选择符比较方便，能够对某种标签定义样式，但是有的时候并不想对网页内所有此标签应用同一种样式，而只想对某些标签应用某种样式，它们不一定是相同的标签。常见格式如下：

```
.类名{ 样式属性 1:值 1; 样式属性 2:值 2; 样式属性 3:值 3; ..... }
```

例如：

```
.center
{
    background-color: white;      //背景颜色
    text-align: center;          //文本水平布局
}
```

如果某个标签的 class 属性设置为类选择符的名称，则类选择符就会对此标签起作用。例如：

```
<h1 class="center">类选择符</h1>
<h2 class="center">类选择符</h2>
```

h1 和 h2 的 class 属性都设置为 center，则 center 样式规则就会对它们起作用。

注意：类名的前面有个“.”，表示是自定义类。利用样式生成器定义的时候，系统会自动加上这个点，但引用的时候不要带点。

### (3) ID 选择符

ID 选择符用于对某一个元素定义样式规则，只能用于某个元素。常见格式如下：

```
#ID 名称{ 样式属性 1:值 1;  样式属性 2:值 2;  样式属性 3:值 3; ..... }
```

例如：

```
#header
{
    background-color: white;           //背景颜色
    text-align: center;               //文本水平布局
}
```

如果某个标签的 ID 属性的值和 ID 选择符的名称相同，则 ID 选择符会对此标签生效。

例如：

```
<div id="header"> ... </div>
```

可以看出，类选择符和 ID 选择符的定义方式非常相似，但也存在一定的区别。多个标签可以使用同一个自定义类，而 ID 却只能用于某一个标签。

利用 CSS 定义样式时还要注意，如果某个元素同时受标签选择符、类选择符和 ID 选择符的影响，则要注意它们的优先级问题：ID 选择符的优先级最高，其次是类选择符，标签选择符的优先级最低。

(4) 伪类选择符

“伪类”是特殊的类，能自动地被支持 CSS 的浏览器所识别。伪类的一个最常用的应用是指定超链接标签<A>，使用伪类可以为访问过的、未访问过的、激活的及鼠标指针悬停于其上的 4 种状态的超链接定义不同的显示样式，即

```
A:link           //代表未访问过的超链接
A:visited        //代表访问过的超链接
A:hover          //当鼠标指针移动到超链接上
A:active         //当超链接处于选中状态
```

A 是超链接标签。在 CSS 定义中，A:hover 必须被置于 A:link 和 A:visited 之后才有效，而 A:active 必须被置于 A:hover 之后才有效。伪类名称对大小写不敏感。例如：

```
a:active
{
    color: blue;           //超链接被选中时的颜色
    background-color: buttonface; //超链接被选中时的背景颜色
}
```

(5) 包含选择符

包含选择符用于定义具有层次关系的样式规则，它由多个样式选择符组成，一般格式如下：

```
选择符 1 选择符 2...{ 样式属性 1:值 1;样式属性 2:值 2; 样式属性 3:值 3; ..... }
```

各选择符之间用空格分割。例如：

```
DIV P H1
{
    font-weight: bold;
```

```
        color: red;
    }
}
```

只有处于 DIV 标签内的 P 标签内的 H1 标签才会应用如上的样式，其他地方的 H1 不会受此样式影响。

**【例 1-3】** 用 CSS 美化【例 1-1】中的登录网页页面。

设计步骤如下：

(1) 新建一个命名为“example1-3”的文本文件，把下面的代码添加到 example1-3.txt 文件中并保存。

```
<HTML>
<HEAD>
<Title>简单的 CSS 应用示例</Title>
<style type="text/css" >
    p {font-family:宋体;font-size:14pt;color: Blue }
    td {font-size: 12px; background-color:Silver}
    a:link {color: #FF0000}           // 未访问的链接颜色
    a:visited {color: #00FF00}        // 已访问的链接颜色
    a:hover {color: #FF00FF}         // 鼠标移动到链接上时的颜色
    a:active {color: #0000FF}        // 选定的链接颜色
</style>
</HEAD>
<BODY >
<form name="form1" method="post" id="form1">
    <p align="center">用户登录</p>
    <table align="center" border="1" width="40%">
        <tr>
            <td align="right">用户名: </td>
            <td><input name="TextBoxName" type="text" id="TextBoxName" /></td>
        </tr>
        <tr>
            <td align="right">密码: </td>
            <td><input name="TextBoxPwd" type="text" id="TextBoxPwd" /></td>
        </tr>
    </table>
    <p align="center">
        <input type="submit" name="BtnOK" value="确定" id="BtnOK" />
        <a href="http://www.baidu.com">百度</a></p>
</form>
</BODY>
</HTML>
```

(2) 更改 example1-3.txt 的后缀名，改成 example1-3.html。用浏览器打开它，浏览器会对文档中的不同标记进行解释并显示指定的效果，打开后的网页页面如图 1.4 所示。

在【例 1-3】的代码中，<style></style>标签对括起来的部分就是 CSS 样式表。其中：

(1) p {font-family:宋体;font-size:14pt;color: Blue} 中的“p”代表该样式是针对<p>标签

的。font-family 表示字体，font-size:14pt 表示字体大小，color: Blue 表示字体颜色。



图 1.4 CSS 应用示例页面

(2) td {font-size: 12px; background-color:Silver}中的“td”代表该样式是针对<td>标签的。font-size:12px 表示字体大小，background-color:Silver 表示背景颜色。

### 1.2.4 脚本语言

HTML 提供了较完善的设计页面的功能，但它提供的信息大多是静态的，这些信息被下载到客户计算机后，就是固定不变的，一些原本可以在客户端完成的任务（如数据合法性检查等），不得不依靠服务器来完成，这样既加重了服务器的负担，也增加了网络的负载。于是，Netscape 公司与 Sun 公司于 1995 年合作开发了 JavaScript 脚本语言，弥补了上述不足，大大提高了客户端的交互性。在此之后，微软公司也发布了自己的脚本语言 VBScript。例如：

```
<script type="text/javascript">this.location.href='http://baidu.com'</script> //使用 JavaScript 链接到百度
<script type="text/vbscript" >MsgBox("Hello,World!")</script> //使用 VBScript 弹出
                                "Hello,World!"对话框
```

脚本语言是介于 HTML、Java、C++ 和 Visual Basic 之类的编程语言之间的语言。HTML 通常用于格式化文本和链接网页，而编程语言则通常用于向计算机发送一系列复杂指令。脚本语言也可用来向计算机发送指令，但它们的语法和规则没有可编译的编程语言那样严格和复杂。脚本语言主要用于格式化文本和使用以编程语言编写的已编译好的组件。

无论哪种脚本语言，其运行环境只局限于 Web 浏览器。它们只能通过 Web 浏览器去完成某种操作，而不是像普遍意义上的程序那样可以独立运行。因为需要由 Web 浏览器进行解释和执行，所以脚本语言并不像其他编译型程序设计语言那样用途广泛。

**【例 1-4】** 利用 JavaScript 脚本语言检查登录用户名是否为空值，若为空值则弹出对话框提示。

设计步骤如下：

(1) 新建一个命名为“example1-4”的文本文件，把下面的代码添加到 example1-4.txt 文件中并保存。

```
<HTML>
<HEAD>
<Title>简单的 HTML 示例</Title>
<script type="text/javascript">
function demo()
{
    if (document.form1.TextBoxName.value=="")
```

```

        { alert("请输入用户名!");}
    }
</script>
</HEAD>
<BODY>
<form name="form1" method="post" id="form1">
    <p align="center">用户登录</p>
    <table align="center" border="1" width="40%">
        <tr>
            <td align="right">用户名: </td>
            <td><input name="TextBoxName" type="text" id="TextBoxName" /></td>
        </tr>
        <tr>
            <td align="right">密码: </td>
            <td><input name="TextBoxPwd" type="text" id="TextBoxPwd" /></td>
        </tr>
    </table>
    <p align="center">
        <input type="submit" name="BtnOK" value="确定" id="BtnOK" onclick="demo();"/></p>
    </form>
</BODY>
</HTML>

```

(2) 更改 example1-4.txt 的后缀名, 改成 example1-4.html。用浏览器打开它, 不输入用户名并单击【确定】按钮, 浏览器弹出对话框并提示“请输入用户名”, 如图 1.5 所示。



图 1.5 JavaScript 示例

以上网页源文件中, `<script>`和`</script>`标签括起来的部分就是 JavaScript 代码。代码中只包含了一个名为 `demo` 的函数, 此函数的功能是判断用户名是否为空, 若为空则弹出一个对话框, 显示“请输入用户名”文字信息。在网页的“确定”按钮中, 通过设置其 `onclick` 事件, 使用户单击该按钮时调用 `demo` 函数。

### 1.2.5 HTTP协议与URL

WWW (World Wide Web, 万维网) 简称 Web, 是 Internet (又称国际互联网) 上最方便、最受用户欢迎的信息服务类型。WWW 上集中了全球的信息资源, 是存储和发布信息的地方, 也是人们查询信息的场所。Internet 中包含了成千上万的 WWW 服务器。

Web 浏览器和服务器用超文本传输协议 HTTP (HyperText Transfer Protocol) 来传输 Web 文档, 通过统一资源定位符 URL (Uniform Resource Locator) 标识文档在网络上服务器的位

置及服务器中的路径。

### 1. HTTP协议

客户端与服务器要进行正常的通信，必须遵循统一的传输内容和传输协议，否则将无法正常通信。在 Web 世界里，传输协议使用的是 HTTP。传输的内容为 HTML 超文本标记语言。有了这两项标准，Web 浏览器便知道如何向 Web 服务器发送请求，而 Web 服务器也知道如何将请求的资源传送到 Web 浏览器，如图 1.6 所示。

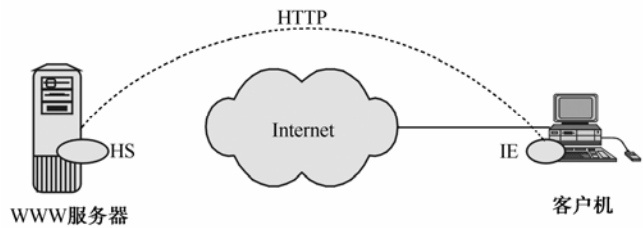


图 1.6 传输 Web 文档

例如：

```
http://www.baidu.com //通过http协议打开百度主页
```

因为 WWW 服务器使用的主要协议是 HTTP 协议，故可以在浏览器中省略 “http://”。

例如：

```
www.baidu.com //省略 “http://”
```

HTTP 协议的主要特点可概括如下：

- (1) 支持客户/服务器（C/S）模式。
- (2) 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD 和 POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。
- (3) 灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。
- (4) 无链接：无链接的含义是限制每次链接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开链接。采用这种方式可以节省传输时间。
- (5) 无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次链接传送的数据量增大。另一方面，当服务器不需要先前信息时其应答就较快。

### 2. URL

WWW 的信息分布在全球，要找到所需信息就必须有一种说明该信息存放在哪台计算机的哪个路径下的定位信息。统一资源定位器 URL 就是用来确定某信息位置的方法。

URL 的概念实际上并不复杂，就像指定一个人要说明他的国别、地区、城镇、街道、门牌号一样，URL 指定 Internet 资源要说明它位于哪台计算机的哪个目录中。URL 通过定义资源位置的抽象标识来定位网络资源，其格式如下：

对于 Internet，“信息服务类型”是指 Internet 的协议名，包括 ftp（文件传输服务）、http（超文本传输服务）、gopher（Gopher 服务）、mailto（电子邮件地址）、telnet（远程登录服务）、news（提供网络新闻服务）和 wais（提供检索数据库信息服务）。

“信息资源地址”指定一个网络主机的域名或 IP 地址。在有些情况下，主机域名后还要加上端口号，域名与端口号之间用冒号“:”隔开。这里的端口是指操作系统用来辨认特定信息服务的软件端口。一般情况下，WWW 服务器程序采用标准的保留端口号（80），因此，用户在 URL 中可以省略。以下是 URL 的一些例子：

http://www.whitehouse.gov:70	//使用 http 协议，“70”是端口号
telnet://www.sina.com	//telnet 是 Internet 的远程登录协议
ftp://ftp.w3.org/pub/www/doc	//使用文件传输协议
gopher://gopher.internet.com	//Internet 上提供双向沟通的界面（查询系统）
news://comp.sys.novell	//使用网络新闻服务
wais://quake.think.com/directory-of-servers	//使用 wais 协议，提供检索数据库信息服务

注意：以上只是示例，在实际网络中并不可访问。

3. 绝对路径和相对路径

实际应用中，路径通常被分为绝对路径和相对路径两种表示方法。  
所谓绝对路径，就是指带有完整域名的文件路径。例如：

http://www.sohu.com/index.htm	//其中“www.sohu.com”为域名
-------------------------------	-----------------------

所谓相对路径，就是由目标文件与当前文件的相对位置来表示的文件路径。例如：

<a href = "../index.html">上一级 index.html 页面</a>	//链接到上一级目录中的 index.html 页面
---	----------------------------

其中“..”表示上一级目录，“../..”表示上上级的目录，以此类推。  
在实际中还有如下符号：

(1) “.”代表目前所在的目录。例如：

<a href = "./SubDir/Red.gif " >下一级目录的 Red.gif</a>	//打开下一级目录的 Red.gif 图片
---	-----------------------

(2) “/”代表根目录。例如：

<a href = "/Red.gif">打开根目录下的 Red.gif 文件</a>	//打开根目录下的 Red.gif 图片
---	----------------------

【例 1-5】 利用相对路径和绝对路径打开各个目录下的文件。  
设计步骤如下：

(1) 在桌面上新建一个命名为“example1-5”的文件夹，在此文件夹中新建一个命名为“example1-5”的文本文件，把下面的代码添加到 example1-5.txt 文件中并保存。

<html>
<head>
<title>example1-5 相对路径和绝对路径</title>
</head>



```
<body>
<a href = "C:/girl.gif">打开根目录下的 girl.gif 文件</a><br> //利用绝对路径打开
<a href = "../example1-4.html">上一级 example1-4.html 页面</a><br>
<a href = "./SubDir2/boy.gif ">下一级目录的 boy.gif</a><br>
<a href = "/girl.gif">打开根目录下的 girl.gif 文件</a><br> //利用相对路径打开
</body>
</html>
```

(2) 更改 example1-5.txt 的后缀名，改成 example1-5.html。在 C 盘下存放一张 “girl.gif” 图片，把 example1-4.html 复制到桌面上，同时在 example1-5 文件下新建一个命名为“SubDir2”的文件夹并在此文件夹下存放一张 boy.gif 文件。

(3) 用浏览器打开 example1-5.html 文件，分别单击各个链接，浏览器分别打开不同的文件，结果如图 1.7 所示。

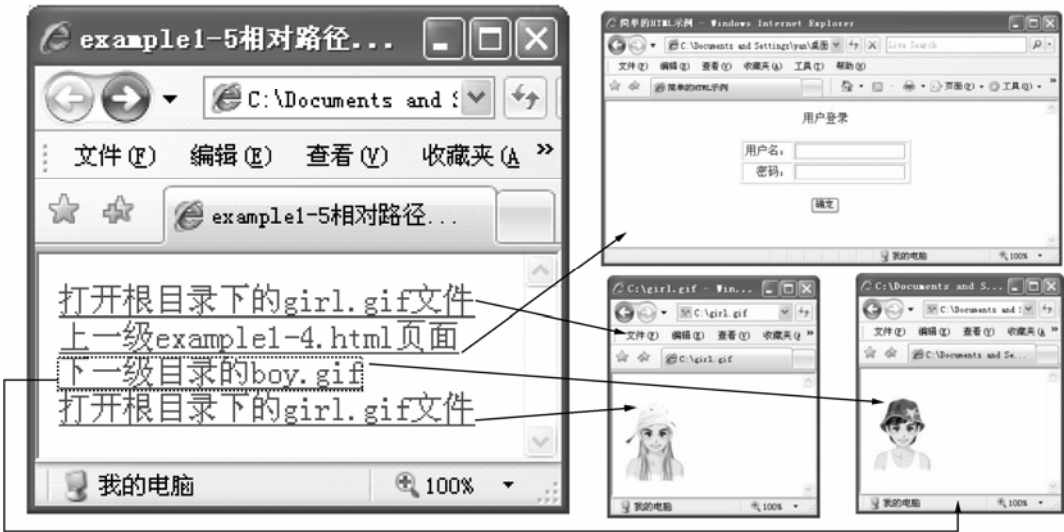


图 1.7 利用相对和绝对路径打开文件

1.2.6 HTML表单与Web请求

HTML 中有一对特殊的标签<form></form>，在这一对标签之间可以包含许多称为“表单控件”的标签，这些表单控件通常是用来获取用户的输入的，而后当单击某个特定的按钮后，所有表单控件中的用户输入的值将被提交到服务器，从而形成一次新的 Web 请求。例如：

```
<form name="form1" action="HTMLPage.htm" method="post" id="form1">
    用户名: <input name="TextBoxName" type="text" id="TextBoxName" />
    <input type="submit" name="BtnOK" value="确定" id="BtnOK"/>
</form>
```

其中包含一个文本框（用于输入用户名）和一个提交按钮。单击【确定】按钮后，数据就会被传送至表单的“action”属性所指向的页面，形成了一次 Web 请求。

常用的表单控件有文本框、密码框、单选钮、复选框、下拉列表框、按钮等。

Web 请求发送的方式有 2 种，分别是 GET 和 POST。上面例子中采用的是 POST 方式，

这是 Web 表单的默认方式。

虽然这两种方式都可以向服务器提交数据，但是其应用的环境还是有区别的。一般情况下，用户在网页表单中输入数据，并单击某个按钮之后发送的请求，通常使用 POST 方式。

而用户通过地址栏输入要访问的资源地址，或者通过网页中的超链接访问资源时，采用的是 GET 方式。例如：

```
http://www.sohu.com/sport/index.aspx?Id=2 //此处的Id参数以GET方式传送到服务器
```

从提交的形式来看，GET 和 POST 方式有很大区别。GET 方式本质上是将用户需要提交的数据转存到 URL 中，作为资源请求的一部分传送到 Web 服务器，当服务器接收到这样的资源请求时，会将其中的用户数据部分从地址中分离出来。

**【例 1-6】** 使用 POST 方式将登录页面的用户名和密码传到 HTMLPage.htm 文件中。

设计步骤如下：

(1) 在桌面上新建一个命名为“example1-6”的文件夹，在此文件夹中分别新建一个命名为“example1-6”和一个命令为“HTMLPage”的文本文件，把下面的代码添加到 example1-6.txt 文件中并保存。

```
<html >
<head>
    <title>form 示例</title>
</head>
<body bgcolor="#ccff66" >
<form name="form1" action="HTMLPage.htm" method="post" id="form1">
    <p align="center">用户登录</p>
    <table align="center" border="1" width="40%">
        <tr>
            <td align="right">用户名: </td>
            <td><input name="TextBoxName" type="text" id="TextBoxName" /></td>
        </tr>
        <tr>
            <td align="right">密码: </td>
            <td><input name="TextBoxPwd" type="text" id="TextBoxPwd" /></td>
        </tr>
    </table>
    <p align="center">
        <input type="submit" name="BtnOK" value="确定" id="BtnOK"/></p>
    </form>
</body>
</html>
```

(2) 把下面的代码添加到 HTMLPage.txt 文件中，更改 example1-6.txt 和 HTMLPage.txt 的后缀名，分别改成 example1-6.html 和 HTMLPage.htm。

```
<html >
<head>
    <title>HTMLPage 页面</title>
```

```
</head>
<BODY>
<h1>这是 HTMLPage 页面</h1>
</body>
</html>
```

(3) 用浏览器打开 `example1-6.html` 文件，输入用户名和密码并单击【确定】按钮，结果如图 1.8 所示。

当单击【确定】按钮后，浏览器打开了 HTMLPage 页面，并把用户名和密码传到 HTMLPage 页面中。至于 HTMLPage 页面是怎样获得数据的，在以后的章节中会为大家介绍。

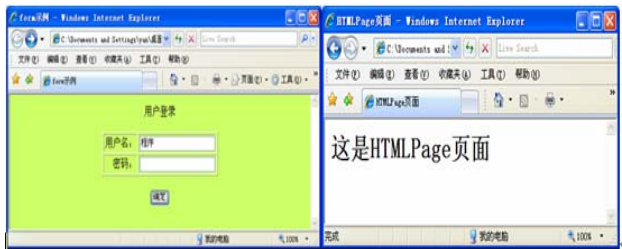


图 1.8 将登录页面的用户名和密码传到 HTMLPage 页面

## 1.3 Web 开发技术的发展

1991 年，CERN（Conseil European pour la Recherche Nucleaire，欧洲原子核研究委员会）正式发布了 Web 技术标准。1993 年出现了第一个 HTML 浏览器，促进了 Internet 的革命性变化。从那时起，Web 开发技术迅猛发展，各种新的技术层出不穷。

### 1.3.1 静态网页与动态网页

早期的 Web 网站以提供信息为主要功能，设计者事先将固定的文字及图片放入网页中，这些内容只能由人工更新，这种类型的页面被称为“静态网页”。静态网页文件的扩展名通常为 `htm` 或 `html`（前面介绍的例子就是静态网页）。

然而，随着应用的不断增强，网站需要与浏览者进行必要的交互，从而为浏览者提供更为个性化的服务。因此 HTML 3.2 提供了一些表现动态内容的标记，本书前面提到的 `<form>` 标签和其他一些表单控件标签就是此类标记。例如，`<input></input>` 标签可以提供一个文本框或按钮。有了这些基本元素，Web 服务器就能通过 Web 请求了解用户的输入操作，从而对此操作做出相应的响应。由于整个过程中页面的内容会随着操作的不同而变化，因此通常将这种交互式的网页称为“动态网页”。

HTML 表单使得 Web 页面设计者可以编写标准的输入页面，但是，这些仍不足以使 HTML 具备足够的可编程能力，这种 Web 设计更像是一种平面设计技术，而不是程序设计技术，设计出来的页面只能呈现静态的文本和图像，无法满足用户需要的交互模式。因此，Web 开发技术不可避免地要向动态技术演化。

### 1.3.2 客户端动态技术

从理论上来说，Web 架构是典型的分布式应用架构。Web 应用中的每一次信息交换都涉及客户端和服务端两个层面。因此，提供动态 Web 页内容的开发技术大体上也可以分为客户端技术和服务器端技术两大类。下面介绍客户端动态技术的发展。

在客户端模型中，附加在浏览器上的模块（如插件）完成创建动态页的全部工作。采用的主要技术如下。

(1) JavaScript: JavaScript 是一种脚本语言，主要控制浏览器的行为和内容。它依赖于内置于浏览器中的被称为脚本引擎的模块。

(2) VBScript: 与 JavaScript 类似，但仅 IE 支持。

(3) ActiveX 控件: ActiveX 控件是一个组件，用高级语言编写，可以嵌入网页并提供特殊的客户端功能，如计时器、条形图、数据库访问、客户端文件访问、网络功能等。ActiveX 控件依赖于浏览器中安装的 ActiveX 插件，IE 默认安装该插件，但 Netscape 需另外安装插件。

(4) Java 小应用程序 (JavaApplet): 与 ActiveX 控件类似，比 JavaScript 的功能更强大，支持跨平台。JavaApplet 依赖于浏览器中安装的 Java 虚拟机 (Java Virtual Machine, JVM) 才能运行。

### 1.3.3 服务器端动态技术

每一个提供动态内容的服务器端技术，都依赖于添加到服务器（而不是客户端）的模块附件，这些模块在服务器中完成创建动态页的全部工作，然后将结果转换为 HTML 传送回浏览器。常用的主要技术如下。

#### 1. CGI

公共网关接口 (Common Gateway Interface, CGI)，是添加到 WEB 服务器的模块，提供了在服务器上创建脚本的机制。CGI 允许用户调用 Web 服务器上的另一个程序（如 Perl 脚本）来创建动态 Web 页，且 CGI 的作用是将用户提供的数据传递给该程序进行处理，以创建动态 Web 应用程序。CGI 可以运行于许多不同的平台（如 UNIX 等）。不过 CGI 存在不易编写、消耗服务器资源较多的缺点。

#### 2. JSP

JSP 页面 (Java Server Pages)，是一种允许用户将 HTML 或 XML 标记与 Java 代码相组合，从而动态生成 Web 页的技术。JSP 允许 Java 程序利用 Java 平台的 JavaBeans 和 Java 库，运行速度比 ASP 快，具有跨平台特性。已有允许用户在 IIS 服务器中使用 JSP 的插件模块。

#### 3. PHP

该技术是指 PHP 超文本预处理程序 (Hyper Text Processor)。它起源于个人主页 (Personal Home Pages)，使用一种创建动态 Web 页的脚本语言，语法类似 C 和 Perl 语言。PHP 是开放源代码和跨平台的，可以在 Windows NT 和 UNIX 上运行。PHP 的安装较复杂，会话管理功能不足。

#### 4. ASP

动态服务器页面（Active Server Pages , ASP），是 Web 服务器上的模块（asp.dll 文件）。它允许使用 VBScript 和 JavaScript 脚本语言编程，在服务器端使用 Windows 提供的任何功能，如数据库存取、E-mail 收发、网络功能、文件处理、图形处理、系统功能等。但它只能在 Windows 平台上运行。

## 5. ASP.NET

ASP.NET 是一种基于 .NET 框架开发动态网页的新技术，它依赖于 Web 服务器上的 ASP.NET 模块（aspnet\_isapi.dll 文件），但该模块本身并不处理所有工作，它将一些工作传递给 .NET 框架进行处理。它允许使用多种面向对象语言编程，如 VB.NET、C#、C++、Jscript.NET 和 J#.NET 语言等。

## 1.4 .NET Framework与ASP.NET

随着 Internet 应用的迅速发展，为了适应用户对 Web 应用持续增长的需要，微软公司于 2002 年正式发布 .NET Framework 和 Visual Studio .NET 开发环境，使之成为一个支持多语言、通用的运行平台，并在其中引入了全新的 ASP.NET Web 开发技术。

### 1.4.1 .NET Framework

从 2002 年正式发布 .NET Framework 1.0 后，.NET Framework 不断更新，2008 年已正式发布 .NET Framework 3.5 版本，同时也提供了 Visual Studio 2008（以下简称 VS2008）集成开发环境。

#### 1. .NET Framework的结构

.NET Framework 是一个多语言组件开发和运行环境，它提供了一个跨语言的统一编程环境。.NET Framework 的目的之一是为了让开发人员更容易地建立 Web 应用程序和 Web 服务，使 Internet 上的各应用程序之间可以使用 Web 服务进行沟通。开发人员可以将远端应用程序提供的服务和单机应用程序的服务结合在一起，组成一个整体的应用程序。

目前，.NET Framework 主要包括如下内容。

（1）.NET 语言：五种基本语言的编译器，包括 C#、Visual Basic、J#（Java 语言的克隆体）、具有托管扩展的 C++ 及 Jscript .NET（JavaScript 的服务器端版本）。

（2）.NET FCL（Framework Class Library，框架类库）：包括对 Windows 和 Web 应用程序、数据访问、Web 服务等方面的支持。

（3）CLR（Common Language Runtime，公共语言运行库）：.NET Framework 核心的面向对象引擎，可以执行所有的 .NET 程序，并且为这些程序提供自动服务，如安全检测、内存管理及性能优化等。.NET Framework 的结构如图 1.9 所示。

#### 2. .NET应用程序的编译过程

实际上，由于 CLR 将所有代码先编译成 MSIL（MicroSoft Intermediate Language，微软中间语言），然后再由 JIT（Just In Time）编译器编译成本机机器语言代码，因此，理论上 .NET 可以应用在 UNIX、Linux、Mac OS 或其他操作系统上。而且由于 CLR 的存在，使得 .NET

Framework 消除了异类框架之间的差别，所以也可以让开发人员选择其喜好的编程语言。上述处理过程如图 1.10 所示。

对于 ASP.NET 应用程序，使用 MSIL 和 JIT 技术还能够提高执行效率。当第一次执行 ASP.NET 程序时，它被先编译为 MSIL，再由 JIT 编译器将 MSIL 编译为机器码，并将机器码存放在缓存中。以后再执行该程序时，只要程序没有变化，系统将直接从缓存中读取机器码，从而大大提升了执行效率。

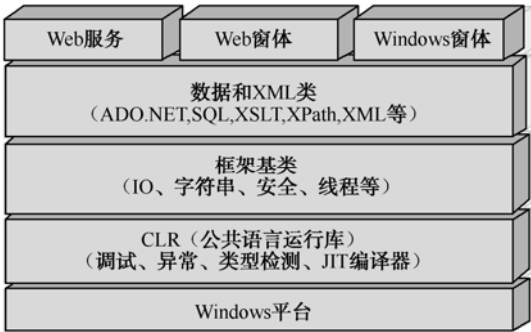


图 1.9 .NET Framework 的结构

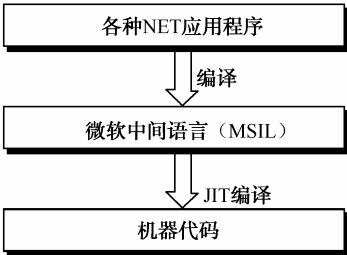


图 1.10 .NET 应用程序的编译过程

## 1.4.2 ASP.NET

### 1. ASP.NET与ASP的区别

ASP.NET 沿袭了 ASP 的名称，但是在实质上已经完全超越了 ASP，不是 ASP 的简单升级，而是全新一代的动态网页实现系统。ASP.NET 是微软发展的新体系结构.NET 的一部分，是 ASP 和 .NET 技术的结合；提供基于组件、事件驱动的可编程网络表单，大大简化了编程。ASP.NET 还可以用来建立网络服务。

### 2. ASP.NET 3.5 的设计目标

在 ASP.NET 3.5 的设计过程中，微软公司深入市场，针对大量开发人员和软件公司的需求，进行了卓有成效的研究。这些细致的工作，为 ASP.NET 3.5 的开发打下了良好的基础。同时，也为 ASP.NET 3.5 这个产品设定了宏伟目标。ASP.NET 3.5 设计目标的核心就是“简化”。围绕这个核心，建立了以下几个主要设计目标。

(1) 提高生产效率：设计目标是将应用程序代码减少约 70%，以提高开发效率，使开发人员可以将更多的注意力放到如何实现业务逻辑上。对于部分开发工作，尽量做到不写或少写代码就可完成任务。

(2) 提高性能和可伸缩性：在 ASP.NET 1.x 中，应用程序性能和可伸缩性一直是两个饱受争议的话题。ASP.NET 3.5 在这两个方面进行了重大改进，主要体现在优化内部处理机制、扩展高速缓存功能及增加对 64 位服务器的支持等方面。

(3) 提高对不同终端设备的支持能力：优化 ASP.NET 3.5 技术，提高对浏览器、PDA、移动电话等终端设备的支持能力，并且增加对各种未来终端设备的支持。另外，通过扩展控件功能，使得同一控件能够输出支持多种设备的代码。

(4) 简化管理和维护工作：在 ASP.NET 1.x 中，Web 应用程序的管理和维护主要通过人

工方式，即对 XML 配置文件进行处理来完成，这种方式容易出错。在 ASP.NET 3.5 中，目标是创建图形化的管理工具，简化管理和维护工作。

### 3. ASP.NET 3.5 的编译系统

在 ASP.NET 1.x 中进行编译是一个困难的过程，如果对后台编码文件进行了修改，则这些修改的内容不能反映到应用程序中，除非重新建立整个应用程序，也就是说，在重新编译整个应用程序之前，必须再次逐个地请求每个页面。

ASP.NET 3.5 在处理类和编译的方式方面都进行了改进。由于 .NET Framework 3.5 支持使用局部类（Partial Class）的功能，因此 ASP.NET 3.5 提供了一个与 ASP.NET 1.x 不同的后台编码模型。在编译时，各个文件将组合为一个产品，这将生成非常简洁的后台编码页面。类中页面代码和后台编码类是分开的。而在 ASP.NET 1.x 中与此相反，页面需要从它自己的后台编码文件中派生，以表示一个逻辑页面。

ASP.NET 3.5 应用程序中可以包含一个 \App\_Code 目录，其中放置类的源代码，这些类都是动态编译的，并且在应用程序中反映出来。在进行修改时，不需要像 ASP.NET 1.x 那样使用独立的建立过程，这是一个“只要保存了就可以使用”的部署模型。同时 VS 2008 也为 \App\_Code 目录中的对象提供了 IntelliSense。

ASP.NET 3.5 还提供了一些工具，可以预先编译 ASP.NET 应用程序，包括.aspx 页面和后台编码，这样第一次访问页面时，不会出现延迟现象。如果页面中有错误，则即使不调用每个页面，也可以找出这些错误。还可以预先编译整个应用程序，再将生成的程序集部署到服务器，从而防止代码在部署后被窃取、篡改和损坏。

## 1.5 ASP.NET应用程序的组成

一个成功发布的 ASP.NET 应用程序通常包括以下 6 个部分。

- (1) 一个在 IIS 信息服务器中的虚拟目录。这个虚拟目录被配置为应用程序的根目录。
- (2) 一个或多个带.aspx 或.ascx 扩展名的文件。
- (3) Web.config 应用程序配置文件。
- (4) Global.asax 全局文件。
- (5) 保留文件夹，用于系统特定类型的文件。

(6) bin 目录，包含发布网站时生成的若干程序集（.dll 文件），这些程序集通常是在应用程序中引用的控件、组件或其他代码。应用程序将自动引用此目录的代码所表示的任何类。要求此目录位于 Web 应用程序的根目录下。

### 1.5.1 虚拟目录

虚拟目录又称为目录的“别名”，它是以服务器作为根的目录。默认安装时，IIS 的 Web 服务器主目录被设置为 C:\inetpub\wwwroot，该目录对应的 URL 是 http://localhost/。在因特网中向外发布信息或接受信息的应用程序必须放在虚拟目录或其子目录下面。系统将自动在虚拟目录下去寻找相关的文件。

将应用程序放在虚拟目录下，有两种方法。

- (1) 直接将网站的根目录放在虚拟目录下。例如，应用程序的根目录是“myweb”，直接

将它放在虚拟目录下，物理路径为“C:\Inetpub\wwwroot\myweb”。此时对应的 URL 是“http://localhost/myweb”。

(2) 将应用程序目录放到一个物理目录下（例如，D:\myweb），同时建一个虚拟目录指向该物理目录，此时新建的虚拟目录名只是一个别名，并不要求与被指向的物理目录同名。客户只需要通过虚拟目录的 URL 来访问即可，并不需要知道对应的物理目录在哪里。这样做的好处是一旦应用程序的物理目录有了改变时，只需更改目录映射，无须更改虚拟目录名，客户仍然可以用原 URL 来访问它们。

### 1.5.2 网页文件

网页是 Web 应用程序运行的主体。ASP.NET 中的基本网页以 aspx 作为后缀。除此以外，应用程序中还可以包括以 ascx 为后缀的用户控件，即以传统的 html 或 asp 为后缀的网页。

当服务器打开后缀为 htm 的网页时，服务器将不经过任何处理就直接送往浏览器。而当服务器打开后缀为 aspx 的网页时，需先运行服务器端的代码，然后再将结果转换成 HTML 的代码形式送往浏览器。对于曾经请求过而又没有改变的 aspx 网页，服务器会直接从缓冲区中取出结果而不需要再次运行。

因此，对于一个即使不包含服务器端代码的 HTML 网页，也允许使用 aspx 作为文件的后缀。此时服务器会解读此网页，当它发现其中并不包括服务器端代码时，也会将文本送往浏览器，其他什么事情也不做，其结果只是稍微降低了程序的运行效率。因此尽管允许纯 HTML 网页使用“.aspx”后缀，但并不提倡这样做。反过来，如果网页中包括有服务器控件或服务器端代码，而仍然采用“.htm”后缀，则将会出现错误。

### 1.5.3 网站配置文件（Web.config文件）

Web.config 是一个基于 XML 语言的配置文件。该文件的作用是对 Web 应用程序进行配置，如规定客户的认证方法、基于角色的安全技术的策略、数据绑定的方法、错误显示方式、数据库连接串等。

Web.config 并不是网站必备的文件。因为服务器有一个总的配置文件，名为“Machine.config”，默认安装在“C:\windows\Microsoft.NET\Framework\（版本号）\CONFIG\”的目录下。这个配置文件已经确定了所有 ASP.NET 应用程序的基本配置，通常情况下不要去修改这个文件，以免影响其他应用程序的正常运行。

### 1.5.4 网站全局文件（Global.asax文件）

Global.asax 文件也是一个可选的文件，但是一个应用程序最多只能建立一个 Global.asax 文件，而且必须放在应用程序的根目录下。这是一个全局性的文件，用来处理应用程序级别的事件，放置如 Application\_Start、Application\_End、Application\_Error 和 Session\_Start、Session\_End 等事件的处理代码。

当应用程序运行的时候，Global.asax 的内容被编译到一个继承自 HttpApplication 类的类中。因此，HttpApplication 类中的所有的方法、类和对象对于应用程序都是可用的。

CLR 监控着 Global.asax 的变化。如果它察觉到这个文件发生了改变，那么将自动启动一个新的应用程序复本，同时创建一个新的应用程序域。原应用程序域当前正在处理的请求被允许结束，而任何新的请求都交由新应用程序域来处理。当原应用程序域的最后一个请求处



理完成时，这个应用程序域即被清除。这有效地保证了应用程序可以重新启动，而不被任何用户察觉。

注意：为防止应用程序用户下载应用程序而看到源代码，ASP.NET 默认配置为阻止用户查看 Global.asax 的内容。如果有人在浏览器中输入 URL “http://localhost/progaspnet/Global.asax”，则将会收到一个 403（禁止访问）的错误信息。

### 1.5.5 保留文件夹

ASP.NET 使用应用程序根目录下的许多特殊目录来维护应用程序的内容和数据。ASP.NET 3.5 在 ASP.NET 1.x 中已存在的 Bin 目录的基础上又引进了另外 7 个受保护的目录，这些目录也不是一定要存在的，每个目录需要开发者在需要时手动创建或通过 VS2008 创建。

最常用的 3 个保留目录如下。

(1) App\_Code: 包含页使用的类的源文件 (.cs 或.vb)。所有的文件必须使用相同的语言。

(2) App\_Data: 包含应用程序数据文件，包括 MDF 文件、XML 文件和其他数据存储文件。ASP.NET 3.5 使用此目录来存储应用程序的本地数据库。

(3) App\_Themes: 包含应用程序支持的主题和外观的定义，也是保留目录中唯一一个可以通过 HTTP 请求访问的目录。

## 1.6 VS 2008 集成开发工具简介

在 ASP.NET 1.x 时代，开发人员主要使用 Visual Studio .NET 2003 开发工具。随着 ASP.NET 3.5 的发布，与之相适应的工具也产生了，这就是 Visual Studio 2008 (VS 2008)。相对于之前的版本，VS 2008 提供了更好的集成开发环境，可高效地创建各种类型的.NET 应用程序或组件，如 Windows 应用程序、XML Web Service、.NET 组件、移动应用程序、ASP.NET 应用程序等。与之前版本相同，也默认支持多种编程语言，如 C#、VC++、VB.NET、VJ#等。除此之外，VS 2008 还提供了许多新特性，能够帮助不同类型的开发人员快速创建各类应用程序。VS 2008 集成开发环境如图 1.11 所示。

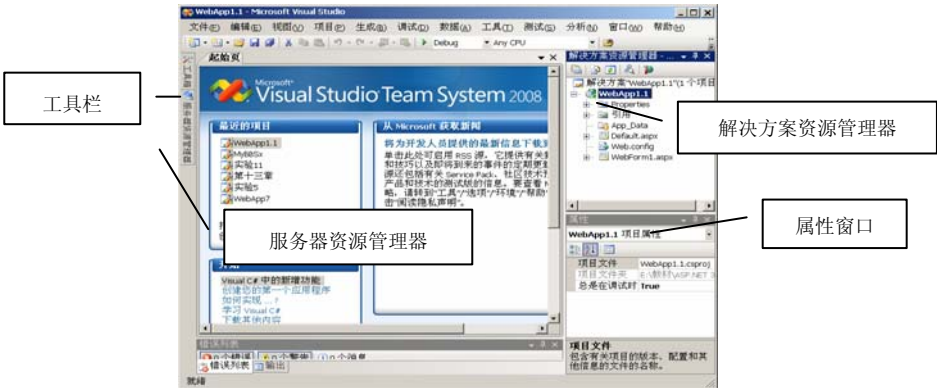


图 1.11 VS 2008 集成开发环境

## 1.6.1 编译和运行Web应用程序

应用程序编写完成后就可以编译和运行了，在 VS 2008 中编译网站很简单，只需单击“生成”菜单中的“生成网站”选项即可，或右击网站项目，在弹出的快捷菜单中选择“生成网站”选项。

运行应用程序也很简单，只需单击“调试”菜单中的“开始执行（不调试）”选项即可运行当前网页，或右击项目中的 `aspx` 网页文件，在弹出的快捷菜单中选择“在浏览器中查看”选项。

下面简要说明一下应用程序编译和运行的一些相关知识。

编译应用程序的目的是为了给用户提供请求服务。`ASP.NET` 编译引擎首先将代码编译成一个或多个程序集，即扩展名为 `.dll` 的文件。该文件可以采用多种不同的语言编写，如 `VB.NET`、`C#`、`J#` 等。编译时，编译引擎将代码翻译成 `MSIL`（微软中间语言），`MSIL` 是一种与编程语言和 `CPU` 无关的表示形式。运行时，`MSIL` 将运行在 `.NET Framework` 的上下文中，`.NET Framework` 会将 `MSIL` 编译成 `CPU` 特定的指令，以便计算机处理器运行应用程序。

编译应用程序主要有四个好处。

### （1）提高执行速度

经过编译的代码的执行速度比脚本语言快得多，因为它是更接近机器代码的表示形式，并且不需要进行其他分析。

### （2）增强安全性

编译后的代码比源代码更难进行反向工程处理，因为编译后的代码缺乏高级语言所具有的可读性和抽象性。此外，模糊处理工具增强了编译后的代码对抗反向工程处理的能力。

### （3）有利于检查和消除错误

在编译代码时，将自动检查代码的语法错误、类型安全问题及其他问题。通过在生成时捕获这些问题，可以及时消除代码中的许多错误。

### （4）提高互操作性

由于 `MSIL` 代码支持任何 `.NET` 语言，因此，可以在代码中使用其他语言编写的程序集。例如，正在用 `C#` 语言编写 `ASP.NET` 程序时，可以添加对使用 `VB.NET` 编写的 `.dll` 文件的引用。

## 1.6.2 部署Web应用程序

所谓“部署”是指将源站点文件发布到远程 `Web` 服务器上。`VS 2008` 提供了发布网站的功能，开发人员可以将需要部署的网站发布到某个目录，然后将该目录中的文件直接部署到 `IIS` 中。

发布网站很简单，只需单击“生成”菜单中的“发布网站”选项即可，或右击网站项目，在弹出的快捷菜单中选择“发布网站”选项。图 1.12 所示为“发布网站”对话框。

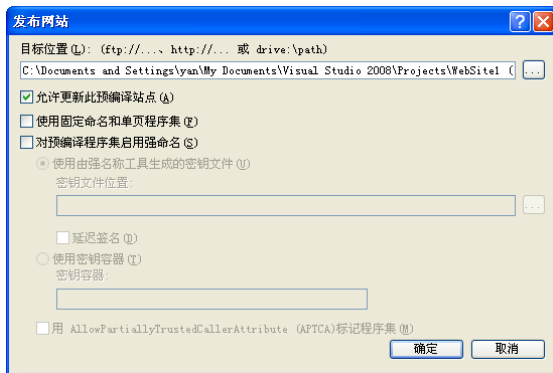


图 1.12 “发布网站”对话框

发布站点工具要求设置发布目标位置、是否允许更新预编译站点、是否使用固定命名和单页程序集，以及是否对预编译程序集启用强命名等选项。

发布站点工具首先对站点进行编译，然后将编译结果输出到目标位置。与直接复制站点文件相比，使用发布站点工具有以下优点。

- (1) 预编译过程能发现任何编译错误、Web.config 及其他文件中的潜在错误。
- (2) 可选择不可更新预编译和可更新的预编译选项，这样不会随站点部署任何程序代码，从而为源文件提供一项安全措施。
- (3) 由于站点中的网页已经编译过，因此在最初请求时无须对其进行动态编译，减少了网页的初始响应时间。

不过，在使用发布站点工具时还需注意三个方面。

- (1) 根据发布选项的不同，在对站点进行更改后可能需要重新编译站点。因此，在开发站点并频繁地更改网页的过程中，使用发布站点工具可能不行。
- (2) 发布站点工具不能将已编译的站点直接部署到远程服务器，只能将其复制到本地计算机或局域网上的一台服务器。
- (3) 发布站点工具会重写目标文件夹中的文件，因此，一定要发布到可以安全删除现有内容的位置。

### 1.6.3 使用帮助系统

VS 2008 的帮助系统较先前的版本有了重大改进，可以更高效地访问帮助内容。VS 2008 的帮助系统由文档资源管理器呈现，如图 1.13 所示。文档资源管理器主要由 3 部分组成，一是菜单和命令按钮，包括文件、工具等菜单，前进、后退、如何实现等命令按钮；二是以选项卡形式集成的目录、索引、帮助收藏夹；三是搜索栏和帮助信息显示的主窗口。



图 1.13 “帮助”界面

从用户界面而言，没有太大变化，但内部进行了很大改进，主要体现在以下几个方面。

(1) 默认情况下，是在文档资源管理器中查看帮助内容，而不是在 VS 2008 集成开发环境中显示。

(2) 增加了“如何实现”按钮设置。该按钮是一个访问帮助系统的新工具，它显示常见开发任务的组织层次结构。对于使用不同程序语言、创建不同类型应用程序的开发人员，该工具提供了数百个任务，且每个任务都包含技术指导内容和示例代码。

(3) 提供社区集成功能。用户可以直接从文档资源管理器访问网上论坛，以便提出问题、搜索感兴趣的线索或查看帖子的状态等。

(4) 增加搜索结果的摘要。每条搜索结果会显示每个主题的摘要，同时，主题摘要的底部还会显示一些图标用来提供附加信息。

(5) 提供帮助筛选器。在目录选项卡中文档树的上方包含一个“筛选依据”下拉框，该下拉框中包括 .NET Framework、Visual C# 等选项，用户可通过更改选项来设置帮助筛选条件。在索引选项卡中也包括一个同样的筛选器，可确定目录和索引选项卡中显示的项的范围。这些筛选器不会对搜索结果产生影响。

(6) 不仅支持搜索本地帮助文件，而且还支持搜索联机帮助内容。联机帮助内容包括在线 MSDN、专注于 .NET 技术的 Codezone 站点、在线论坛等。

## 1.7 简单的 ASP.NET 程序实例

下面使用 VS 2008 来创建一个简单的 ASP.NET 应用程序，实际体验一下 VS 2008 的开发过程和 ASP.NET 的特性。本书程序都是用 C# 语言开发的。

### 1.7.1 创建 Web 应用程序的一般步骤

使用 VS 2008 创建 ASP.NET 应用程序的一般步骤如下。

(1) 创建 Web 项目或网站

启动 VS 2008，依次单击菜单“文件”→“新建”→“网站”，或单击菜单“文件”→“新建”→“项目”并选择“ASP.NET Web 应用程序”模板，然后选择位置路径，即可创建 Web 应用程序项目。

(2) 添加网页文件

单击菜单“项目”→“添加新项”，选择“Web 窗体”模板，然后输入网页文件名称，即可创建一个网页。

(3) 在网页中添加控件

在解决方案资源管理器中选中网页文件，切换到设计视图，从“工具箱”中选择控件拖入页面中，即可完成网页的界面设计。

(4) 编写网页功能代码

在网页中选中控件，双击该控件即可切换到代码设计器，编写相应的事件代码，即可完成网页的功能设计。

(5) 浏览网页

右击项目中的 aspx 网页文件，在弹出的快捷菜单中选择“在浏览器中查看”选项即可浏览选择的网页。

以上是创建单个网页的基本步骤，若要添加多个网页，可重复步骤（2）、（3）、（4）。

(6) 部署应用程序

右击网站项目，在弹出的快捷菜单中选择“发布网站”选项，即可完成部署操作。

1.7.2 一个简单的ASP.NET程序实例

根据前述创建 ASP.NET 应用程序的一般步骤，下面在 VS 2008 编辑环境下设计登录页面。

**【例 1-7】** 设计一个单页 Web 应用程序，网页界面由 1 个表格、1 个标签（Label）、2 个文本框和 1 个命令按钮（Button）组成，另外还包含若干文字。设计后的界面如图 1.14 所示。

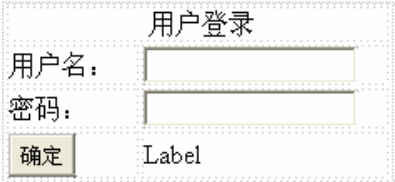


图 1.14 实例设计效果

1. 创建应用程序界面

(1) 创建 Web 应用程序项目

启动 VS 2008，选择“新建”→“项目”，项目类型选择“Web”，模板选择“ASP.NET Web 应用程序”，如图 1.15 所示。然后指定该应用程序的保存路径，确定后系统则新建了一个解决方案（默认名为 WebApplication1），该解决方案中包含一个 Web 项目（默认名为 WebApplication1），该项目中包含一个网页文件（默认名为 Default.aspx），此时屏幕上会出现一个空白窗体（在设计视图下）。

**注意：**ASP.NET 3.5 为每个页面提供了三种编辑视图，一种是设计视图，提供了可视化的所见即所得的开发环境；另一种是源视图，它是 ASP.NET 3.5 的源代码视图；还有一种就是拆分视图，可同时看到设计视图和源视图。

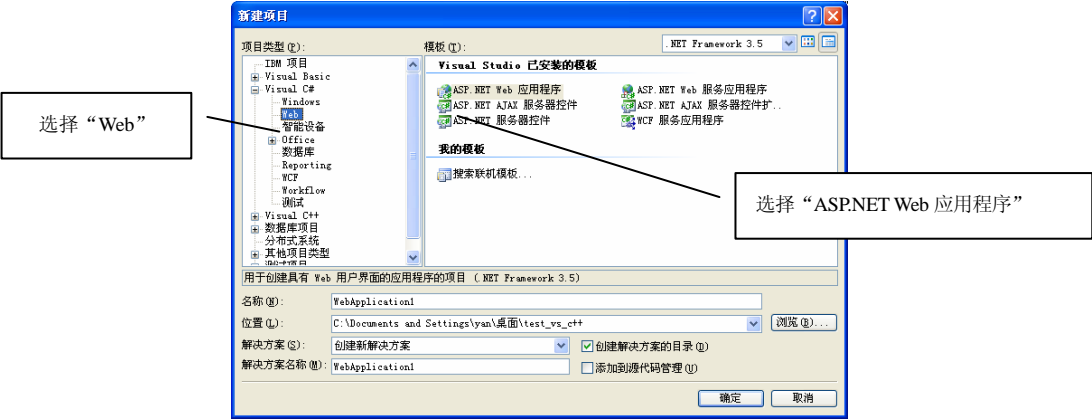


图 1.15 创建 Web 应用程序项目

(2) 向网页中添加控件

将 Default.aspx 改名为 Login.aspx，对于本例界面，首先使用“表”→“插入表”菜单命令，插入一个 4 行 2 列的表格，居中显示。在第 1 行中输入“用户登录”字样，并放入中间位置，然后将工具箱中的标签（Label）、文本框（TextBox）和按钮（Button）等 3 种控件放入表格中：先将光标定位到表格的单元格中，将鼠标移到左侧“工具箱”位置，自动弹出“工具箱”窗口，选择其中的“TextBox”控件对象，然后双击即可在网页中放入一个文本框对象，按类似的操作方法可分别添加标签和按钮。完成后的效果如图 1.16 所示。

2. 设置控件的属性

可以通过“属性”窗口为控件设置属性。单击按钮【Button】，在“属性”窗口中出现按钮“Button1”的所有属性。滚动属性列表，选定属性名“Text”，在右列中输入属性值为“确定”，如图 1.17 所示。另外将输入密码的文本框的 TextMode 属性设置为 Password。

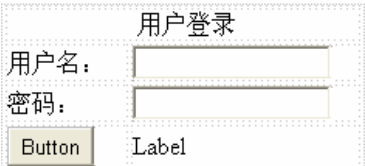


图 1.16 添加标签图

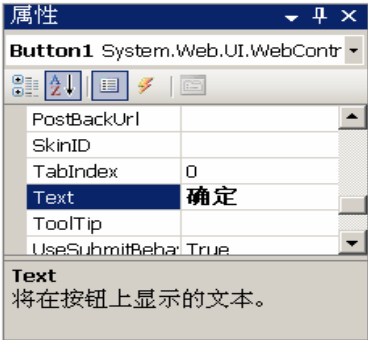


图 1.17 设置按钮属性

3. 编写程序代码


当窗体中控件的属性设置完成后，下一步就要编写代码来实现功能了。本例中有 1 个事件要处理，就是“确定”按钮的鼠标单击事件。VS 2008 的集成开发环境能自动生成事件代码的模板，用户只需在生成的模板中添加自己的代码即可。

为按钮对象“确定”添加鼠标单击事件代码有 2 种方法。

(1) 双击要编写代码的命令按钮，系统自动打开代码编辑器，并出现如下代码行：

```
protected void Button1_Click(object sender, EventArgs e)
{ }
```

双击时在代码编辑器的过程列表框中出现 **Button1\_Click** 事件，它是该控件的默认事件。

(2) 在 **Button** 控件的“属性”窗口中，单击图标，添加 **Click** 事件方法“**Button1\_Click**”，双击则系统自动切换到代码编辑器并生成如方法（1）所示的代码，如图 1.18 所示。也可以直接双击 **Click** 事件所要输入的方法空白处，系统自动生成 **Button1\_Click** 方法。

在事件代码中输入如下的程序代码：


```
protected void Button1_Click(object sender, EventArgs e)
{
    if ((TextBox1.Text == "admin") && (TextBox2.Text == "admin"))//用户名和密码都是 admin
    {   Label1.Text = "欢迎进入论坛系统";   }
    else
    {   Label1.Text = "用户名或密码错！ ";   }
}
```

4. 保存应用程序

使用“文件”菜单中的“全部保存”命令或单击工具栏上的“全部保存”按钮，可以将所有编辑过的代码和设计的网页存盘。

5. 运行和调试程序

运行程序有如下几种方法。

- (1) 右击项目中的 **aspx** 网页文件，在弹出的快捷菜单中选择“在浏览器中查看”选项即可浏览选择的网页。
- (2) 从“调试”菜单中选择“开始执行（不调试）”命令。
- (3) 单击工具栏中的按钮.
- (4) 按【**Ctrl+F5**】组合键。

运行程序后，即显示网页界面，输入用户名“**admin**”、密码“**admin**”，单击【**确定**】按钮，标签中将显示“欢迎进入论坛系统”，如图 1.19 所示。

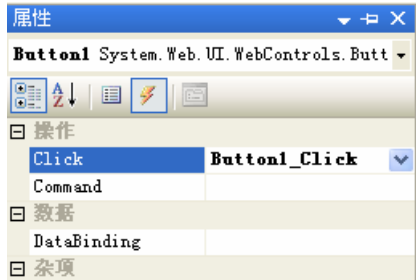


图 1.18 添加事件

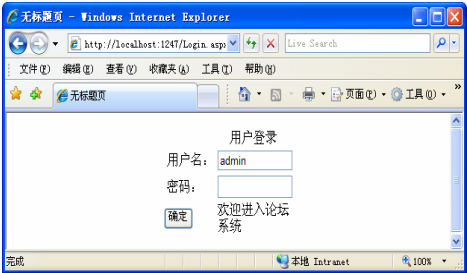


图 1.19 实例运行效果

6. 部署应用程序



在“解决方案资源管理器”中右击项目名称，在弹出的快捷菜单中选择“发布”选项，在如图 1.20 所示的对话框中选择目标位置，本例选择“D:\MyWeb”，单击【发布】按钮即可将网站代码发布到目标文件夹中。目标文件夹中的文件结构如图 1.21 所示，其中的 bin 目录中包含一个程序集文件“WebApp1.1.dll”。目标文件夹即是最终的网站应用程序，可以将其直接部署（即复制）到 Web 服务器中。

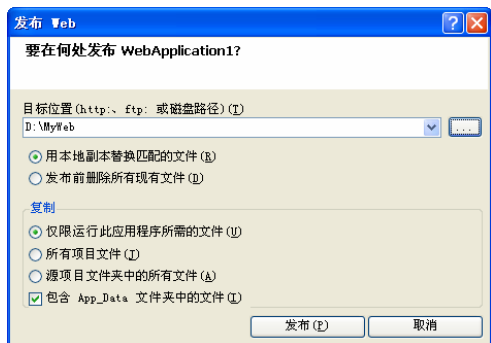


图 1.20 发布项目

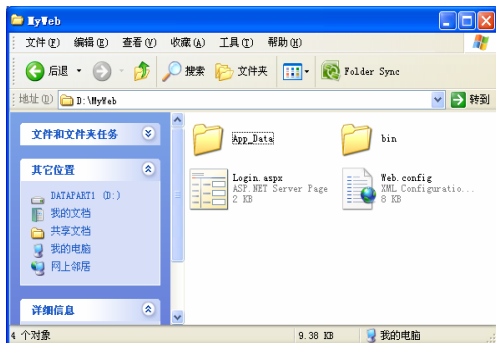


图 1.21 目标文件夹中的文件结构

## 习 题

1. 阐述 Web 工作原理。
2. 什么是 HTML?
3. URL 是什么?
4. JavaScript 脚本语言是一种什么语言?
5. CSS 的作用是什么?
6. 客户端动态技术有哪些?
7. 服务器端动态技术有哪些?
8. 如何发布 Web 应用程序?
9. .NET Framework 主要包括哪些内容?
10. CLR 的作用是什么?
11. 开发 ASP.NET 应用程序，必须具有的工具包括（ ）。  
A. .NET Framework      B. VS 2008      C. IIS      D. 数据库系统
12. 如何用 VS 2008 新建一个 Web 项目或网站?





写操作就直接在这块内存区域进行。例如：

```
int iNum=10;           // 分配一个 32 位内存区域给变量 iNum，并将 10 放入该内存区域
iNum=iNum+10;          // 从变量 iNum 中取出值，加上 10，再将计算结果赋给 iNum
```

图 2.2 给出了上述值类型的操作示意。

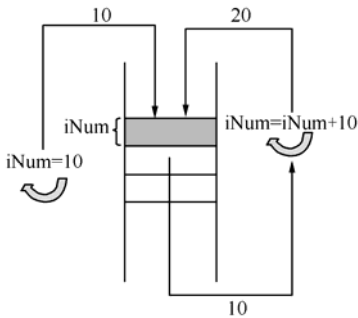


图 2.2 值类型操作示意图

简单类型是系统预置的，一共有 13 个数值类型，如表 2.1 所示。

表 2.1 C#简单类型

C#关键字	.NET CTS 类型名	说 明	范 围 和 精 度
short	System.Int16	16 位有符号整数类型	-327 68~327 67
ushort	System.UInt16	16 位无符号整数类型	0~65535
int	System.Int32	32 位有符号整数类型	-214 748 364 8~214 748 364 7
uint	System.UInt32	32 位无符号整数类型	0~429 496 729 5
long	System.Int64	64 位有符号整数类型	-922 337 203 685 477 580 8~922 337 203 685 477 580 7
ulong	System.UInt64	64 位无符号整数类型	0~184 467 440 737 095 516 15
char	System.Char	16 位字符类型	所有的 Unicode 编码字符
float	System.Single	32 位单精度浮点类型	$\pm 1.5\times 10^{-45} \sim \pm 3.4\times 10^{38}$ (大约 7 个有效十进制数位)
double	System.Double	64 位双精度浮点类型	$\pm 5.0\times 10^{-324} \sim \pm 3.4\times 10^{308}$ (大约 15~16 个有效十进制数位)
bool	System.Boolean	逻辑值（真或假）	true, false
decimal	System.Decimal	128 位高精度十进制数类 型	$\pm 1.0\times 10^{-28} \sim \pm 7.9\times 10^{28}$ (大约 28~29 个有效十进制数位)
sbyte	System.SByte	8 位有符号整数类型	-128~127
byte	System.Byte	8 位无符号整数类型	0~255

表中的“C#关键字”是指在 C#中声明变量时可使用的类型说明符。例如：

```
int myNum           // 声明 myNum 为 32 位的整数类型
```

.NET 的 CTS 包含所有简单类型，它们位于.NET 框架的 System 名字空间。C#的类型关

键字就是.NET 中 CTS 所定义类型的别名。从表 2.1 可见，C#的简单数据类型可以分为整数类型、字符类型、实数类型和布尔类型。

注意：C#中的变量必须在声明及初始化之后方可使用，缺一不可。如果仅仅将变量声明，而未将其初始化，则将导致无法使用此变量，程序在编译时便会抛出错误。

【例 2-1】 设计一个网站，命名为 Chapter2，添加一个网页，命名为 example2-1，用于输出所定义的值类型。

设计步骤如下：

(1) 在 D 盘新建一个命名为“Chapter2”的文件夹，打开 VS 2008，依次展开工具栏中“新建”→“网站”选项，系统弹出“新建网站”对话框。在该对话框中，模板选择“ASP.NET 网站”，语言选择“Visual C#”，框架选择“.NET Framework 3.5”，位置选择“D:\Chapter2”。单击【确定】按钮，如图 2.3 所示。

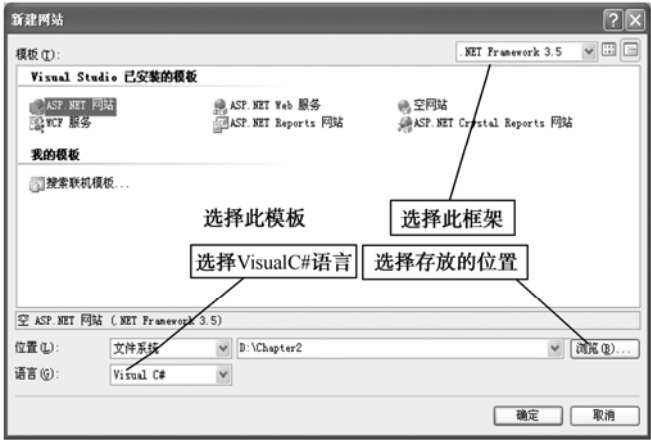


图 2.3 “新建网站”对话框

(2) 打开资源管理器→右击“D:\Chapter2”（如图 2.4 所示）→单击“添加新项”选项，系统弹出“添加新项”对话框，选择“Web 窗体”并且重新命名为“example2-1”。如图 2.5 所示，单击【添加】按钮。

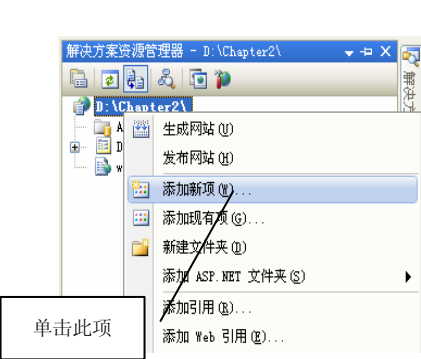


图 2.4 添加新项



图 2.5 添加 Web 窗体

(3) 打开资源管理器→打开刚添加的“example2-1.aspx.cs”文件，如图 2.6 所示。在“Page\_Load”方法中添加如下代码，按【Ctrl+F5】组合键运行网页，结果如图 2.7 所示。

```
int myNum = 100;
myNum = myNum - 10;
Response.Write("myNum 的值为: "+myNum); // 向浏览器发送数据
```

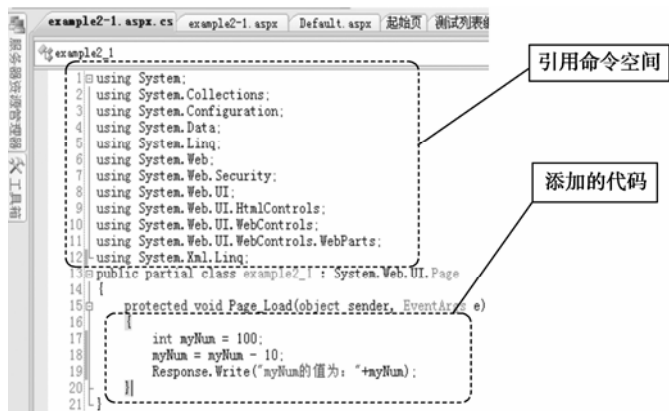


图 2.6 代码页



图 2.7 运行后的结果页面

在 example2-1.aspx.cs 代码页中，系统自动引用一些命名空间，如“using System”。命名空间是为了防止相同名字的不同标识符发生冲突而设计的隔离机制。“Response”是 ASP.NET 内置对象，该对象用于向浏览器发送数据，数据以 HTML 的格式发送。

2.2.2 引用类型

引用类型的变量不存储它们所代表的实际数据，而是存储实际数据的引用。引用类型分两步创建：首先在堆栈上创建一个引用变量，然后在堆上创建对象本身，再把这个内存的句柄（也是内存的首地址）赋给引用变量。例如：

```
string s1, s2;
s1="ABCD"; s2 = s1;
```

其中，s1、s2 是指向字符串的引用变量，s1 的值是字符串“ABCD”存放在内存的地址，这就是对字符串的引用。两个引用变量之间的赋值，使得 s2、s1 都是对“ABCD”的引用，如图 2.8 所示。

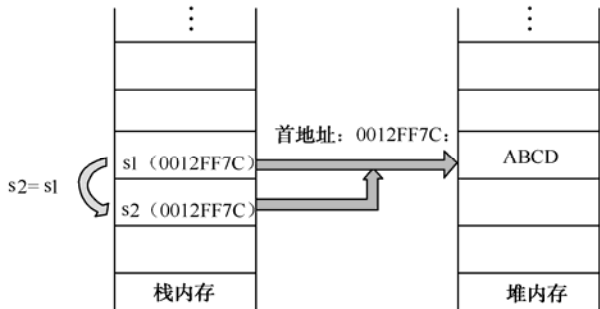


图 2.8 引用类型赋值示意

引用类型包括 class（类）、interface（接口）、数组、delegate（委托）、object 和 string。

其中 `object` 和 `string` 是两个比较特殊的类型。在 C# 的统一类型系统中，所有类型（预定义类型、用户定义类型、引用类型和值类型）都是直接或间接从 `object` 继承的。可以将任何类型的值赋给 `object` 类型的变量。例如：

```
int a = 10;
object abj = a;                                //将 int 类型 a 赋给 object 类型 abj
```

`string` 类型表示 Unicode 字符的字符串。`string` 是 .NET Framework 中 `System.String` 的别名。尽管 `string` 是引用类型，但定义相等运算符是为了比较 `string` 对象（而不是引用）的值。这使得字符串相等性的测试更为直观。例如：

```
string myString1 = "中国";                    //把字符串“中国”赋给字符型变量 myString1
string myString2 = "中国";                    //把字符串“中国”赋给字符型变量 myString2
Console.WriteLine(myString1 == myString2);    //相等，显示 True
```

上面第 3 行代码执行后将显示“True”。当使用“==”直接对 2 个字符串变量进行比较时，系统将比较其字符串内容。

字符串可以使用两种形式表达，即直接使用双引号括起来或者在双引号前加上@。通常情况下，会直接使用双引号来表示字符串，例如：

```
string myString = "sample"。                  //正确
```

但是，当字符串中包含某些特殊转义符时，@ 将非常有用。请看下面出现错误的例子：

```
string myString = "C:\Windows";              //错误
```

因为“\”被认为是转义符的开始，而“\W”却不是系统内置的转义符，因此编译出错。下面是 2 个正确的例子：

```
string myString = "C:\\Windows";              //正确，“\\”转义为“\”
string myString = @"C:\Windows";             //正确
```

对于使用“@”开始的字符串标识，编译器将忽略其中的转义符，而将其直接作为字符处理。

**【例 2-2】** 在网站 Chapter2 上添加一个网页，命名为 `example2-2`，用于字符串引用类型比较。

设计步骤如下：

- （1）在 Chapter2 网站上添加一个命名为“example2-2”的网页。
- （2）在“example2-2.aspx.cs”代码页中添加如下代码：

```
string myString1 = "C:\\Windows";
string myString2 = @"C:\Windows";
string myString3 = myString1;
Response.Write(myString3 + "<br>");           //<br>是 HTML 标签，表示换行
Response.Write(myString1 == myString2);
```

- （3）按 **【Ctrl+F5】** 组合键运行网页，结果如图 2.9 所示。

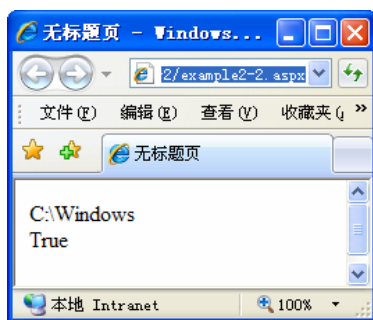


图 2.9 example2-2 网页运行的结果

### 2.2.3 装箱和拆箱

值类型与引用类型之间的转换被称为装箱与拆箱。装箱与拆箱是 C# 类型系统的核心。通过装箱与拆箱操作，可以轻松实现值类型与引用类型的相互转换，任何类型的值最终都可以按照对象来处理。

装箱是值类型转换为 object 类型，或者转换为由值类型所实现的任何接口类型。把一个值类型的值装箱，也就是创建一个对象并把这个值赋给该对象。以下是一个装箱的代码：

```
int i = 1;           //把“1”赋给 int 型变量 i
object obj = i;      //装箱操作
```

拆箱操作正好相反，是从 object 类型转换为值类型，或者将一个接口类型转换为一个实现该接口的值类型。

拆箱的过程分为 2 个步骤，一是检查对象实例是否是给定的值类型的装箱值，二是将值从对象实例中复制出来。下面列出一个简单的拆箱操作代码：

```
int i = 1;           //把“1”赋给 int 型变量 i
object obj = i;      //装箱操作
int j = (int)obj;     //拆箱操作
```

由于拆箱的过程中要对引用类型内存中的值进行检查，因此对于不兼容的类型转换将抛出异常。例如，以下代码会导致 InvalidCastException 异常的出现：

```
int i = 1;           //把“1”赋给 int 型变量 i
object obj = i;      //装箱操作
string j = (string)obj; //类型不兼容
```

虽然装箱与拆箱会带来性能上的损失，但是对于使用者来说，这样做的好处是可以使用相同的方式去对待值类型和引用类型。然而，对于装箱与拆箱所产生的性能损失也不能忽略。在没有必要使用此功能的情况下，应尽量避免使用。

**注意：**当一个装箱操作把值类型转换为一个引用类型时，不需要显式地强制类型转换；而当拆箱操作把引用类型转换为值类型时，由于它可以强制转换到任何可以相容的值类型，所以必须显式地强制类型转换。

**【例 2-3】** 在网站 Chapter2 上添加一个网页，命名为 example2-3，用于体现装箱和拆箱操作。

设计步骤如下：

- (1) 在 Chapter2 网站上添加一个命名为 “example2-3” 的网页。
- (2) 在 “example2-3.aspx.cs” 代码页中添加如下代码：

```
int x = 123, y;  
object obj1=x;           // 装箱操作  
x = x+100;                // 改变 x 的值，此时 obj1 的值并不会随之改变  
y = (int) obj1;           // 拆箱操作，必须进行强制类型转换  
Response.Write(" x= " + x + "<br>"); //x=223  
Response.Write(" y= " + y);       //y=123
```

- (3) 按【Ctrl+F5】组合键运行网页，结果如图 2.10 所示。

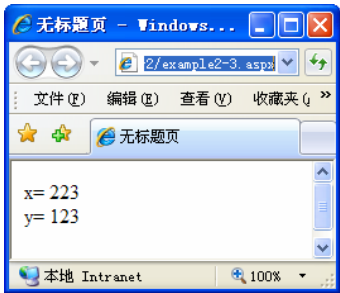


图 2.10 example2-3 网页运行的结果

## 2.3 常量和变量

无论使用何种程序设计语言编写程序，变量和常量都是构成一个程序的基本元素，我们可以从它们的定义、命名、类型、初始化等几个方面来认识、理解变量和常量。

### 2.3.1 常量

常量，顾名思义就是在程序运行期间其值不会改变的量。常量及其使用非常直观，以能读懂的固定格式表示固定的数值，每一种值类型都有自己的常量表示形式。例如：

```
168U           // 这是一个 uint 类型常量  
32L            // 这是一个 long 类型常量
```

常量通常可以分为以下几种。

#### 1. 整型常量

对于一个整数值，默认的类型就是能保存它的最小整数类型，其类型可以分为 int、uint、long 或 ulong。如果默认类型不是想要的类型，则可以在常量后面加后缀 (U 或 L) 来明确指定其类型。在常量后面加 L 或 l（不区分大小写）表示长整型。例如：

```
32             // 这是一个 int 类型  
32L            // 这是一个 long 类型
```

在常量后面加 U 或 u（不区分大小写）表示无符号整数。例如：

```
128U           // 这是一个 uint 类型
128UL          // 这是一个 ulong 类型
```

整型常量既可以采用十进制也可以采用十六进制，不加特别说明时默认为十进制，在数值前面加 0x（或 0X）则表示十六进制。十六进制基数用 0~9、A~F（或 a~f）表示，例如：

```
0x20           // 十六进制数 20，相当于十进制数 32
0x1F           // 十六进制数 1F，相当于十进制数 31
```

2. 浮点常量

一般带小数点的数或用科学计数法表示的数都被认为是浮点数，其数据类型默认为 double 类型，但也可以加后缀符表明三种不同的浮点格式数。

- (1) 在数字后面加 F（f）表示是 float 类型。
- (2) 在数字后面加 D（d）表示是 double 类型。
- (3) 在数字后面加 M（m）表示是 decimal 类型。

例如：

```
3.14, 3.14e2, 0.168E-2    // 都是 double 类型常量，3.14e2 相当于  $3.14 \times 10^2$ ，0.168E-2 相当于  $0.618 \times 10^{-2}$ 
3.14F, 0.168f             // 都是 float 类型常量
3.14D, 0.168d             // 都是 double 类型常量
3.14M, 0.168m             // 都是 decimal 类型常量
```

3. 字符常量

字符常量，简单地说，就是用单引号括起来的单个字符，如'A'，它占 16 位，以无符号整型数的形式存储该字符所对应的 Unicode 代码。这对于大多数图形字符是可行的，但对一些非图形的控制字符（如回车符）则行不通，所以字符常量的表达有若干种形式。

- (1) 单引号括起来的一个字符，如'A'。
- (2) 十六进制的换码系列，以“\x”或“\X”开始，后面跟 4 位十六进制数，如'\X0041'。
- (3) Unicode 码表示形式，以“\U”或“\u”开始，后面跟 4 位十六进制数，如'\U0041'。
- (4) 显式转换整数字符代码，如(char) 65。
- (5) 字符转义系列，如表 2.2 所示。

表 2.2 字符转义系列

转义字符	含 义	Unicode 码	转义字符	含 义	Unicode 码
\'	单引号	\u0027	\b	退格符	\u0008
\"	双引号	\u0022	\f	走纸换页符	\u000C
\\	反斜线字符	\u005C	\n	换行符	\u000A
\0	空字符	\u0000	\r	回车符	\u000D
\a	警铃符	\u0007	\t	水平制表符	\u0009
\v	垂直制表符	\u000B			



#### 4. 字符串常量

字符串常量是用双引号括起来的零个或多个字符序列。C#支持两种形式的字符串常量，一种是常规字符串，另一种是逐字符字符串。

(1) 常规字符串。双引号括起来的一串字符，可以包括转义字符。例如：

```
"Hello, World\n"           // 表示字符串 Hello,World 加换行符
"C:\\windows\\Microsoft"   // 表示字符串 C:\windows\Microsoft
```

(2) 逐字符字符串。在常规字符串前面加一个@，就形成了逐字符字符串，它的意思是字符串中的每个字符均表示本意，不使用转义字符。如果在字符串中需用到双引号，则可连写两个双引号来表示一个双引号。例如：

```
@ "C:\\windows\\Microsoft" // 与 "C:\\windows\\Microsoft" 含义相同
@ "He said""Hello"" to me" // 与 "He said\\Hello\\ to me" 含义相同
```

#### 5. 布尔常量

它只有两个值：true 和 false。

#### 6. 符号常量

在声明语句中，可以声明一个标识符常量，但必须在定义标识符时就进行初始化，并且定义之后不能再改变该常量的值。具体格式为

```
const 类型 标识符=初值
```

例如：

```
const double PI=3.14159           //标识符 PI 就是 3.14159
```

**【例 2-4】** 在网站 Chapter2 上添加一个网页，命名为 example2-4，用于输出各种类型的常量。设计步骤如下：

- (1) 在 Chapter2 网站上添加一个命名为“example2-4”的网页。
- (2) 在“example2-4.aspx.cs”代码页中添加如下代码：

```
Response.Write(22+"<br>");           //在页面上输出 int 类型常量
Response.Write(22L+"<br>");           //在页面上输出 long 类型常量
Response.Write(238U+"<br>");           //在页面上输出 uint 类型常量
Response.Write(238UL+"<br>");           //在页面上输出 ulong 类型常量
Response.Write(0x20 + "<br>");           //在页面上输出十六进制类型常量
Response.Write(3.14e2 + "<br>");           //在页面上输出 double 类型常量
Response.Write(0.168E-2 + "<br>");           //在页面上输出 double 类型常量
Response.Write(3.14M + "<br>");           //在页面上输出 decimal 类型常量
Response.Write("C:\\windows\\Microsoft"+"<br>"); //在页面上输出字符串类型常量
const double PI = 3.14159;               //声明标识符常量
```

(3) 按【Ctrl+F5】组合键运行网页，结果如图 2.11 所示。

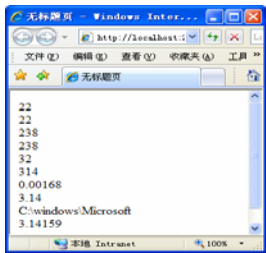


图 2.11 example2-4 网页运行的结果

2.3.2 变量

变量是在程序运行过程中其值可以改变的量，它是一个已命名的存储单元，通常用来记录运算的中间结果或保存数据。在 C#中，每个变量都具有一个类型，它确定哪些值可以存储在该变量中。创建一个变量就是创建这个类型的实例，变量的特性由类型来决定。

C#中的变量必须先声明后使用。声明变量包括变量的名称、数据类型，必要时指定变量的初始值。声明变量的形式为

```
类型 标识符;
或
类型 标识符[=初值][, ...];
```

标识符必须以字母或者\_（下划线）开头，后面跟字母、数字和下划线的组合。例如：

```
name、_Int、Name、x_1 //都是合法的标识符
```

C#是大小写敏感的语言，name、Name 分别代表不同的标识符，在定义和使用时要特别注意。另外变量名不能与 C#中的关键字相同，除非标识符是以@作为前缀的。例如：

```
int    x ; // 合法
float  y1=0.0, y2 =1.0, y3 ; // 合法，变量说明的同时可以设置初始数值
string char // 不合法，因为 char 是关键字
string @char // 合法
```

C#允许在任何模块内部声明变量，模块开始于“{”，结束于“}”。每次进入声明变量所在的模块时，则创建变量并分配存储空间；离开这个模块时，则销毁变量并收回分配的存储空间。实际上，变量只在这个模块内有效，所以称为局部变量，模块区域就是变量的作用域。

**【例 2-5】** 在网站 Chapter2 上添加一个网页，命名为 example2-5，用于定义变量并且赋值，输出各变量的值。

设计步骤如下：

- (1) 在 Chapter2 网站上添加一个命名为“example2-5”的网页。
- (2) 在“example2-5.aspx.cs”代码页中添加如下代码：

```
string name,sex="男"; //声明变量 name,sex, 并且给 sex 赋值
int age; //声明变量 age
name = "王明"; //给 name 赋值
```

age = 27; //给 age 赋值

Response.Write("姓名: "+name+"<br>性别: "+sex+"<br>年龄: "+age); //在页面上输出各个变量值

(3) 按【Ctrl+F5】组合键运行网页，结果如图 2.12 所示。



图 2.12 example2-5 网页运行的结果

2.4 运算符和表达式

表达式是由操作数和运算符构成的。操作数可以是常量、变量、属性等；运算符指示对操作数进行什么样的运算。因此，也可以说表达式就是利用运算符来执行某些计算并且产生计算结果的语句。例如：

int a=3,b=5,c;

c=a+b; //赋值表达式语句，结果是 c 等于 8

C#提供大量的运算符，按需要操作数的数目来分，有一元运算符（如++）、二元运算符（如+，\*）、三元运算符（如?: ）。按运算功能来分，基本的运算符可以分为以下几类：

- ① 算术运算符；
- ② 关系运算符；
- ③ 逻辑运算符；
- ④ 赋值运算符；
- ⑤ 条件运算符；
- ⑥ 位运算符；
- ⑦ 其他（分量运算符 ' . '，下标运算符 '[' ]'等）。

本节主要介绍前 5 种运算符及这些运算符的优先级、结合性等。

2.4.1 算术运算符

算术运算符作用的操作数类型可以是整型也可以是浮点型，如表 2.3 所示。

表 2.3 算术运算符

运 算 符	含 义	示例（假设 x, y 是某一数值类型的变量）	运 算 符	含 义	示例（假设 x, y 是某一数值类型的变量）
+	加	x + y;    x+3;	%	取模	x % y;    11%3; 11.0 % 3;
-	减	x - y;    y-1;	++	递增	++x;    x++;
*	乘	x * y;    3*4;	--	递减	-- x;    x--;
/	除	x / y;    5/2;    5.0/2.0;			

其中：

(1) “+、-、\*、/ ” 运算与一般代数意义及其他语言相同。但需要注意，当 “/” 作用  
的两个操作数都是整型数据类型时，其计算结果也是整型。例如：

```
4/2           // 结果等于 2
5/2           // 结果等于 2
5/2.0         // 结果等于 2.5
```

(2) “%” 为取模运算，即获得整数除法运算的余数，所以也称取余。例如：

```
11%3          // 结果等于 2
12%3          // 结果等于 0
11.0%3        // 结果等于 2，这与 C/C++不同，它也可作用于浮点类型的操作数
```

(3) “++” 和 “—”（递增和递减运算符）是一元运算符，它作用的操作数必须是变量，  
不能是常量或表达式。它既可出现在操作数之前（前缀运算），也可出现在操作数之后（后缀  
运算），前缀和后缀有共同之处，也有很大区别。例如：

```
++x           //先将 x 加一个单位，然后再将计算结果作为表达式的值
x++           //先将 x 的值作为表达式的值，然后再将 x 加一个单位
```

不管前缀还是后缀，它们操作的结果对操作数而言都是一样的，即操作数都加了一个单  
位，但它们出现在表达式运算中是有区别的。例如：

```
int x,y;
x=5; y=++x;    // x 和 y 的值都等于 6
x=5; y=x++;    // x 的值是 6，y 的值是 5
```

2.4.2 关系运算符

关系运算符用来比较两个操作数的值，运算结果为布尔类型的值（true 或 false），如  
表 2.4 所示。

表 2.4 关系运算符

运 算 符	操 作	结果（假设 x，y 是某相应类型的操作数）
>	x>y	如果 x 大于 y，则为 true，否则为 false
>=	x>=y	如果 x 大于等于 y，则为 true，否则为 false
<	x<y	如果 x 小于 y，则为 true，否则为 false
<=	x<=y	如果 x 小于等于 y，则为 true，否则为 false
==	x==y	如果 x 等于 y，则为 true，否则为 false
!=	x!=y	如果 x 不等于 y，则为 true，否则为 false

在 C#中，值类型和引用类型都可以通过==或!=来比较它们的数据内容是否相等。对值类  
型来说，比较的是它们的数据值；而对引用类型来说，由于其内容是对象实例的引用，所以

若相等，则说明这两个引用指向同一个对象实例，如果要测试两个引用对象所代表的内容是否相等，则通常会使用对象本身所提供的方法，如 Equals()。

如果操作数是 string 类型的，则在下列两种情况下被视为两个 string 值相等。

(1) 两个值均为 null。

(2) 两个值都是对字符串实例的非空引用，这两个字符串不仅长度相同，并且每一个对应的字符位置上的字符也相同。例如：

```
int x=1, y=1;
object b1, b2, b3;
string s1="ABCD", s2="1234", s3="ABCD";
b1 = x; b2 = b1; b3 = y;
x == y;           // 结果为 true
b1 ==b2;          // 结果为 true
b1!=b3;           // 结果为 true
s1 ==s2;          // 结果为 false
s1 ==s3;          // 结果为 true
```

关系比较运算“>、>=、<、<=”是以顺序作为比较的标准的，所以它要求操作数的数据类型只能是数值类型，即整型数、浮点数、字符及枚举等类型。

布尔类型的值只能比较是否相等，不能比较大小。因为 true 和 false 值没有大小之分，例如，表达式“true > flase”在 C#中是没有意义的。

2.4.3 逻辑运算符

逻辑运算符是用来对两个布尔类型的操作数进行逻辑运算的，运算的结果也是布尔类型，如表 2.5 所示。

表 2.5 逻辑运算符

运 算 符	含 义	运 算 符	含 义
&	逻辑与	&&	短路与
	逻辑或		短路或
^	逻辑异或	!	逻辑非

假设 p、q 是两个布尔类型的操作数，表 2.6 给出了逻辑运算真值表。

表 2.6 逻辑运算真值表

P	q	p & q	p   q	p ^ q	! p
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	flase	flase	false	false	true

运算符“&&”和“||”的操作结果与“&”和“|”一样，但它们的短路特征使代码的效率更高。所谓短路就是在逻辑运算的过程中，如果计算第一个操作数时就能得知运算结果，则不会再计算第二个操作数。例如：

```

int x,y;
bool z;
x = 1; y = 0;
z = (x > 1) & (++y > 0);           // z 的值为 false, y 的值为 1
z = (x > 1) && (++y > 0);          // z 的值为 false, y 的值为 0

```

逻辑非运算符“!”是一元运算符，它对操作数进行“非”运算，即真/假值互为非（反）。

**【例 2-6】** 在网站 Chapter2 上添加一个网页，命名为 example2-6，用于体现算术运算、关系运算和逻辑运算。

设计步骤如下：

- (1) 在 Chapter2 网站上添加一个命名为“example2-6”的网页。
- (2) 在“example2-6.aspx.cs”代码页中添加如下代码：

```

int myInt1=5,myInt2=0,myInt3=0;
bool myBool1,myBool2;
myBool1 = (myInt1%3 > 1)|(++myInt2 > 0); // “|” 没有短路特性，执行了++myInt2，值为 1
Response.Write("myBool1 为: "+myBool1+"<br>myInt2 的值: "+myInt2+"<br>");
myBool2 = (myInt1%3 > 1)||(++myInt3 > 0); // “||” 具有短路特性，++myInt3 没有执行
Response.Write("myBool1 为: "+myBool2+"<br>myInt3 的值: "+myInt3);

```

- (3) 按【Ctrl+F5】组合键运行网页，结果如图 2.13 所示。

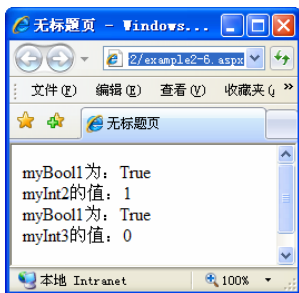


图 2.13 example2-6 网页运行的结果

## 2.4.4 赋值运算符

赋值运算符有两种形式，一种是简单赋值运算符，另一种是复合赋值运算符。

### 1. 简单赋值运算符

语法形式：

```
var = exp
```

赋值运算符左边的称为左值，右边的称为右值。右值是一个与左值类型兼容的表达式（exp），它可以是常量、变量或一般表达式。左值必须是一个已定义的变量或对象（var），因为赋值运算就是将表达式的值存放到左值，因此左值必须是内存中已分配的实际物理空间。例如：

```
int a=1;
int b=++a;           //a 的值加 1 赋给了 b
```

如果左值和右值的类型不一致，则在兼容的情况下需要进行自动转换（隐式转换）或强制类型转换（显式转换）。一般原则是，从占用内存较少的短数据类型向占用内存较多的长数据类型赋值时，可以不做显式的类型转换，C#会进行自动类型转换；反之当从较长的数据类型向占用较少内存的短数据类型赋值时，则必须做强制类型转换。例如：

```
int a=2000;
double b=a;           //隐式转换，b 等于 2000
byte c=(byte)a;       //显示转换，c 等于 208
```

2. 复合赋值运算符

在进行如 `x = x + 3` 的运算时，C#提供了一种简化方式 `x +=3`，这就是复合赋值运算。语法形式：

```
var op= exp           // op 表示某一运算符等价于 var=var op exp
```

除了关系运算符，一般二元运算符都可以与赋值运算符一起构成复合赋值运算，如表 2.7 所示。

表 2.7 复合赋值运算

运 算 符	用 法 示 例	等价表达式	运 算 符	用 法 示 例	等价表达式
+=	x += y	x = x + y	&=	x &= y	x = x & y
-=	x -= y	x = x - y	=	x  = y	x = x   y
*=	x *= y	x = x * y	^=	x ^= y	x = x ^ y
/=	x /= y	x = x / y	%=	x %= y	x = x % y

2.4.5 条件运算符

语法形式：

```
exp1 ? exp2 : exp3
```

其中，表达式 `exp1` 的运算结果必须是一个布尔类型值，表达式 `exp2` 和 `exp3` 可以是任意数据类型，但它们返回的数据类型必须一致。

首先计算 `exp1` 的值，如果其值为 `true`，则计算 `exp2` 的值，这个值就是整个表达式的结果；否则，取 `exp3` 的值作为整个表达式的结果。例如：

```
z = x > y ? x : y;           // z 的值就是 x、y 中较大的一个值
z = x >= 0 ? x : -x;         // z 的值就是 x 的绝对值
```

条件运算符“?:”是 C#中唯一一个三元运算符。

**【例 2-7】** 在网站 Chapter2 上添加一个网页，命名为 `example2-7`，用于体现条件运算符“?:”。

- 设计步骤如下：
- (1) 在 Chapter2 网站上添加一个命名为 “example2-7” 的网页。
  - (2) 在 “example2-7.aspx.cs” 代码页中添加如下代码：

```
int i = 5;
string mystring = --i == 4 ? "i 减 1 后和 4 比较" : "i 和 4 比较后减 1";
Response.Write(mystring);
```

- (3) 按 【Ctrl+F5】 组合键运行网页，结果如图 2.14 所示。

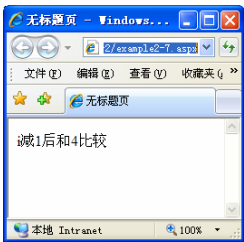


图 2.14 example2-7 网页运行的结果

因为 “--i==4” 为 true，所以返回 “i 减 1 后和 4 比较” 字符串。

2.4.6 运算符的优先级与结合性

当一个表达式含有多个运算符时，C#编译器需要知道先做哪个运算，这就是所谓的运算符的优先级，它控制各个运算符的运算顺序。例如，表达式  $x+5*2$  是按  $x+(5*2)$  计算的，因为 “\*” 运算符比 “+” 运算符的优先级高。

当操作数出现在具有相同优先级的运算符之间时，如表达式 “10-6-2”，按从左到右计算的结果是 2，如果按从右到左计算，则结果是 6。当然 “-” 运算符是按从左到右的顺序计算的，也就是左结合。再如表达式 “x=y=2”，它在执行时是按从右到左运算的，即先将数值 2 赋给变量 y，再将 y 的值赋给 x。所以 “=” 运算符是右结合的。

在表达式中，运算符的优先级和结合性控制着运算的执行顺序，也可以用圆括号 “( )” 显式地标明运算顺序。例如：

$(x+y)*2$	//x 加上 y 后再乘以 2
-----------	-----------------

表 2.8 列出了 C#运算符的优先级与结合性，其中表顶部的优先级较高。

表 2.8 C#运算符的优先级与结合性

类 别	运 算 符	结 合 性
初等项	. ( ) [ ] new typeof checked unchecked	从左到右
一元后缀	++ --	从右到左
一元前缀	++ -- + - ! ~ (T)(表达式)	从右到左
乘法	* / %	从左到右
加法	+ -	从左到右
移位	<< >>	从左到右
关系和类型检测	< > <= >= is as	从左到右



类 别	运 算 符	结 合 性
相等	== !=	从左到右
逻辑与	&	从左到右
逻辑异或	^	从左到右
逻辑或		从左到右
条件与	&&	从左到右
条件或		从左到右
条件	? :	从右到左
赋值	= *= /= %= += -= <<= >>= &= ^=  =	从右到左

2.5 流程控制

一般应用程序代码都不是按顺序执行的，必然要求进行条件判断、循环和跳转等过程，这就需要实现流程控制。在 C#中，主要的流程控制语句包括条件语句、循环语句、跳转语句和异常处理等。

2.5.1 条件语句

条件语句就是条件判断语句，它能让程序在执行时根据特定条件是否成立而选择执行不同的语句块。C#提供两种条件语句结构：if 语句和 switch 语句。

1. if语句

if 语句在使用时可以有几种典型的形式，分别是 if 框架、if\_else 框架、if\_else if 框架及嵌套的 if 语句。

(1) if 框架

语法形式：

```
if(条件表达式) 语句;
```

如果条件为真，则执行语句。这里的语句在语法上是指单个语句，若想执行一组语句，则可将这一组语句用“{”和“}”括起来构成一个块语句，当然在语法上块语句就是一条语句，下面涉及的语句都是这个概念。例如：

```
if (x<0) x = -x ;           // 取 x 的绝对值
if (a+b>c && b+c>a && a+c>b) // 判断数据合法性
{
    p = (a+b+c) / 2 ;
    s = Math.Sqrt (p * (p-a) * (p-b) * (p-c) ); // 求三角形面积
}
```

(2) if\_else 框架

语法格式：

```

if (条件表达式)
    语句 1;
else
    语句 2;

```

如果条件表达式为真，则执行语句 1；否则，执行语句 2。例如：

```

if (a+b>c && b+c>a && a+c>b)           // 判断数据合法性
{
    p = (a+b+c) / 2 ;
    s = Math.Sqrt (p * (p-a) * (p-b) * (p-c) );           // 求三角形面积
}
else
    Console.WriteLine (" 三角形的三条边数据有错! ") ;//输出出错信息

```

### (3) if\_else if 框架

语法形式：

```

if (条件表达式 1)
    语句 1 ;
else if (条件表达式 2)
    语句 2 ;
else if (条件表达式 3)
    语句 3 ;
.....
[ else
    语句 n ; ]

```

这种语句在执行时，从上向下地计算相应的条件表达式，如果结果为真则执行相应语句，跳过 if\_else if 框架的剩余部分，直接执行 if\_else if 框架的下一条语句；如果结果为假，则继续向下计算相应的条件表达式，直到所有的条件表达式都不成立，则执行这个语句的最后部分 else 所对应的语句，或者如果没有 else 语句就什么也不做。例如：

```

if (studentGrade>=90)
    Response.Write ("成绩优秀");
else if (studentGrade>=80)
    Response.Write ("成绩良好");
else if (studentGrade>=60)
    Response.Write ("成绩及格");
else
    Response.Write ("成绩不及格");

```

如果 studentGrade 小于 60 分则输出“成绩不及格”

### (4) 嵌套的 if 语句

在 if 语句框架中，无论条件表达式为真或为假，将要执行的语句都有可能又是一个 if 语句，这种 if 语句又包含 if 语句的结构就称为嵌套的 if 语句。为了避免二义性，C#规定 else 语句与和它处于同一模块的最近的 if 相匹配。例如：

假设有一函数为

$$y = \begin{cases} 1 & (x > 0) \\ 0 & (x = 0) \\ -1 & (x < 0) \end{cases}$$

下面是用嵌套的 if 语句缩写的程序片断。

```
int y=0,x;
if (x>=0)
    if (x>0)
        y=1 ;
    else    y=-1;
```

其中 else 是与最近的 if 匹配的，那么 else 的含义就是 x=0 的情况，所以这个程序在逻辑上是错的，应修正为

```
int y=0,x;
if (x>=0)
{
    if (x>0)y=1 ;
}
else    y=-1;
```

通过对嵌套的 if 语句加“{ }”，把离 else 最近的 if 语句屏蔽了，这样 else 就与 if (x>=0) 匹配，从而正确地完成了这个函数的功能。

2. switch 语句

switch 语句是一个多分支结构的语句，它所实现的功能与 if\_else if 结构相似，但在大多数情况下，switch 语句的表达式更直观、简单、有效。

语法格式：

```
switch (表达式)
{
    case 常量 1:
        语句序列 1;           // 由零个或多个语句组成
        break;
    case 常量 2:
        语句序列 2;
        break;
    .....
    default:                   // default 是任选项，可以不出现
        语句序列 n;
        break;
}
```

switch 语句的执行流程是，首先计算 switch 后的表达式，然后将结果值一一与 case 后的常量值比较，如果找到相匹配的 case，程序就执行相应的语句序列，直到遇到跳转语句

(break), switch 语句执行结束; 如果找不到匹配的 case, 就归结到 default 处, 执行其语句序列, 直到遇到 break 语句为止; 当然如果没有 default, 则不执行任何操作。

C#的 switch 语句需要注意以下几点。

(1) switch 语句的表达式必须是整数类型, 如 char、sbyte、byte、ushort、short、uint、int、ulong、long 或 string、枚举类型。case 常量必须与表达式类型相兼容, case 常量的值必须互异, 不能有重复。

(2) 将与某个 case 相关联的语句序列接在另一个 case 语句序列之后是错误的, 这称为“不穿透”规则, 所以需要跳转语句结束这个语句序列, 通常选用 break 语句作为跳转, 也可以用 goto 转向语句等。“不穿透”规则是 C# 对 C、C++、Java 这类语言中的 switch 语句的一个修正, 这样做的好处是: 一是允许编译器对 switch 语句做优化处理时可自由地调整 case 的顺序; 二是防止程序员不经意地漏掉 break 语句而引起错误。

(3) 虽然不能让一个 case 语句序列穿透到另一个 case 语句序列, 但是可以有两个或多个 case 前缀指向相同的语句序列。

**【例 2-8】** 在网站 Chapter2 上添加一个网页, 命名为 example2-8, 利用 switch 语句把百分制学生成绩换算成等级制成绩并显示在页面上。

设计步骤如下:

(1) 在 Chapter2 网站上添加一个命名为 “example2-8” 的网页。

(2) 从工具栏中拖曳 1 个 Text 控件、1 个 Button 控件和 1 个 Label 控件到页面上。在 Button 控件所对应的属性窗口中, 将 “Text” 属性值 “Button1” 修改为 “转换”。

(3) 双击 Button 控件, 系统自动切换到 “example2-8.aspx.cs” 代码页, 并且添加了 “Button1\_Click” 方法。在 “Button1\_Click” 方法中添加如下代码:

```
int Grade = Convert.ToInt32(TextBox1.Text);           //把string类型的成绩转换为int类型的成绩
switch (Grade / 10)
{
    case 9:
    case 10: Label1.Text = "你的等级制成绩为: A";
        break;
    case 8: Label1.Text = "你的等级制成绩为: B";
        break;
    case 7: Label1.Text = "你的等级制成绩为: C";
        break;
    case 6: Label1.Text = "你的等级制成绩为: D";
        break;
    default: Label1.Text = "你的等级制成绩为: E"; //如果在 60 分以下则为 “E”
        break;
}
```

(4) 按【Ctrl+F5】组合键运行网页, 输入成绩 75, 单击【转换】按钮, 结果如图 2.15 所示。



图 2.15 example2-8 网页运行的结果

## 2.5.2 循环语句

循环语句是指在一定条件下，重复执行一组语句，它是程序设计中的一个非常重要也是非常基本的方法。C#提供了四种循环语句：**while** 语句，**do while** 语句，**for** 语句和 **foreach** 语句。

### 1. while 语句

语法格式：

```
while (条件表达式)
    循环体语句;
```

如果条件表达式为真（**true**），则执行循环体语句。**while** 语句执行流程如图 2.16 所示。  
例如：求 0~100 之间的整数和：

```
int Sum, i;
Sum=0; i=1;
while (i<=100)
{
    Sum+=i;
    i++;
}
Response.Write ("Sum is " + Sum); //在页面上输出结果是：Sum is 5050
```

### 2. do while 语句

语法格式：

```
do
{
    循环体语句 ;
}while (条件表达式);
```

该循环首先执行循环体语句，再判断条件表达式。如果条件表达式为真（**true**），则继续执行循环体语句。**do while** 语句执行流程如图 2.17 所示。

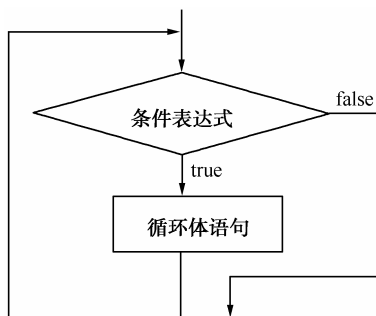


图 2.16 while 语句执行流程

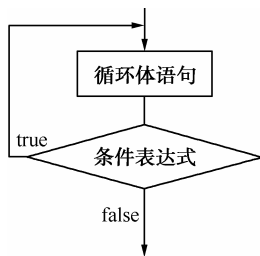


图 2.17 do while 语句执行流程

while 语句与 do while 语句很相似，它们的区别在于 while 语句的循环体有可能一次也不执行，而 do while 语句的循环体至少执行一次。

例如：使用 do while 语句求 0~100 之间的整数和：

```

int Sum=0 , i=1 ;
do
{
    Sum+=i;
    i++;
} while (i<=100);
Response.Write ("Sum is " + Sum);           //在页面上输出结果是：Sum is 5050
  
```

### 3. for 语句

C#的 for 语句是循环语句中最具特色的，它功能较强，灵活多变，使用广泛。

语法格式：

```

for (表达式 1; 表达式 2; 表达式 3)
    循环体语句;
  
```

for 语句执行流程如图 2.18 所示。一般情况下，表达式 1 设置循环控制变量的初值；表达式 2 是布尔类型的表达式，作为循环控制条件；表达式 3 设置循环控制变量的增值（正负皆可）。

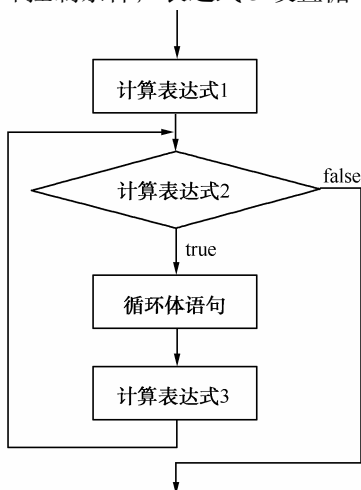


图 2.18 for 语句执行流程图

3 个表达式都是可选的，缺省某个表达式时，其后的分号 “;” 不能省。三个表达式都缺省的情况如下：

```
for(;;){语句}
```

for 语句的执行顺序如下：

- (1) 按书写顺序将表达式 1 执行一遍，为循环控制变量赋初值；
- (2) 测试表达式 2 是否为真；
- (3) 若没有表达式 2 或表达式 2 为真，则执行内嵌语句一遍，按表达式 3 的规律改变循环控制变量的值，回到第二步执行；
- (4) 若表达式 2 不满足，则 for 循环终止。

for 循环具有如下一些变化特点。

(1) for 语句中的表达式 1 和表达式 3 可引入逗号运算符 “,”，这样可以对若干变量赋初值或增值。例如：

```
int Sum , i ;
for (Sum=0,i=1;i<=100;i++)
    Sum+=i;
Response.Write ("Sum is " + Sum);           //在页面上输出结果是： Sum is 5050
for (Sum=0,i=1;i<=100; Sum+=i, i++);       // 循环体是一空语句
Response.Write ("Sum is " + Sum);           //在页面上输出结果是： Sum is 5050
```

(2) for 循环的三个表达式可以任意默认，甚至全部默认，如果表达式 2 默认就约定其值是 true。但不管哪个表达式默认，其相应的分号 “;” 不能默认。例如：

```
int Sum , i ;
for (Sum=0,i=1;i<=100;)                     // 默认表达式 3
    Sum+=i++;
Response.Write ("Sum is " + Sum);           // 在页面上输出结果是： Sum is 5050
for (Sum=0,i=1; ; Sum+=i, i++)              // 默认表达式 2，约定值是 true
    if (i>100) break;                       // 但条件满足时， break 语句跳出循环
Response.Write ("Sum is " + Sum);
Sum=0;i=1;
for ( ; ; )                                 // 三个表达式都默认
{
    Sum+=i++;
    if (i>100)                             // 这种情况一般都会用if语句来设置跳出循环
        break;
}
Response.Write ("Sum is " + Sum);           // 在页面上输出结果是： Sum is 5050
```

(3) 可在 for 循环内部声明循环控制变量。如果循环控制变量仅仅只在这个循环中用到，那么为了更有效地使用变量，也可在 for 循环的初始化部分（表达式 1）声明该变量，当然这个变量的作用域在这个循环内。例如：

```
int Sum =0;
for (int i=1; i<=100; i++)                  //i 只在这个 for 循环中有效
```

```

Sum+=i;
Response.Write ("i = " + i );           // 编译出错, i 这时已经无效
Response.Write ("Sum is " + Sum);        //在页面上输出结果是: Sum is 5050

```

【例 2-9】 在网站 Chapter2 上添加一个网页，命名为 example2-9，利用 for 语句计算百元百鸡问题。所谓百元百鸡问题，就是说公鸡 5 元一只、母鸡 3 元一只、小鸡 1 元 3 只，100 元买 100 只鸡，问公鸡、母鸡、小鸡各多少只？

设计步骤如下：

- (1) 在 Chapter2 网站上添加一个命名为 “example2-9” 的网页。
- (2) 设公鸡数、母鸡数、小鸡数分别为 gj、mj、xj，则要满足如下两个条件：

```

gj+mj+xj=100
5*gj+3*mj+xj/3=100

```

在 “example2-9.aspx.cs” 代码页中添加如下代码：

```

int gj;           //公鸡数
int mj;           //母鸡数
int xj;           //小鸡数
for (gj=0; gj <= 100 / 5; gj++)           //100/5 为 100 元最多可以买到的公鸡数
{
    for (mj=0; mj <= 100 / 3; mj++)           //100/3 为 100 元最多可以买到的母鸡数
    {
        xj = 100 - gj - mj;
        if (xj % 3 == 0 && 5 * gj + 3 * mj + xj / 3 == 100)
        {
            Response.Write("<br>公鸡数: " + gj + "母鸡数: " + mj + "小鸡数: " + xj);
        }
    }
}

```

- (3) 按【Ctrl+F5】组合键运行网页，结果如图 2.19 所示。

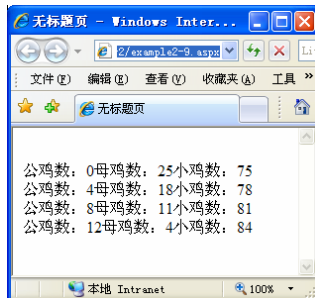


图 2.19 example2-9 网页运行的结果

#### 4. foreach 语句

foreach 语句是 C# 中新引入的，它表示收集一个集合中的各元素，并针对各元素执行内嵌语句。



语法格式:

```
foreach (类型 标识符 in 集合表达式) 语句;
```

其中:

(1) 标识符是指 `foreach` 循环的迭代变量, 它只在 `foreach` 语句中有效, 并且是一个只读局部变量, 也就是说在 `foreach` 语句中不能改写这个迭代变量。其类型应与集合的基本类型相一致。

(2) 集合表达式是指被遍历的集合, 如数组。

在 `foreach` 语句执行期间, 迭代变量按集合元素的顺序依次将其内容读入。

**【例 2-10】** 在网站 Chapter2 上添加一个网页, 命名为 `example2-10`, 利用 `foreach` 语句查找字符串中有多少个字母 “a”。

设计步骤如下:

(1) 在 Chapter2 网站上添加一个命名为 “`example2-10`” 的网页。

(2) 在 “`example2-10.aspx.cs`” 代码页中的 “`Page_Load`” 方法中添加如下代码:

```
int m=0;
string mystring = "laskdjflasdkjasdalfakeoflkdsa";
foreach(char mychar in mystring)
{
    if (mychar == 'a')           //判断迭代变量 mychar 是否为 a 字符
        m++;
}
Response.Write("字符串中有"+m+"个 a");
```

(3) 按 **【Ctrl+F5】** 组合键运行网页, 结果如图 2.20 所示。

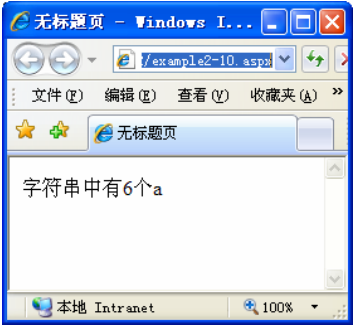


图 2.20 example2-10 网页运行的结果

### 2.5.3 跳转语句

跳转语句用于改变程序的执行流程, 转移到指定之处。C#中有 4 种跳转语句: `continue` 语句、`break` 语句、`return` 语句和 `goto` 语句。它们具有不同的含义, 用于特定的上下文环境中。

#### 1. continue语句

语法形式:

```
continue ;
```

`continue` 语句只能用于循环语句中，其作用是结束本轮循环，不再执行余下的循环体语句，对 `while` 和 `do while` 结构的循环，在 `continue` 执行之后，就立刻测试循环条件，以决定循环是否继续下去；对 `for` 结构的循环，在 `continue` 执行之后，先求表达式 3（即循环增量部分），然后再测试循环条件。通常它会和一个条件语句结合起来使用，而不会是独立的一条语句，也不会是循环体的最后一条语句，否则没有任何意义。例如：

```
for (int n =1; n<=100; n++)
{
    if (n % 3 !=0 )
        continue ;           // 如果 n 不能被 3 整除，则直接进入下一轮循环
    Response.Write (n + " " ); // 只有能被 3 整除的数，才会执行到此，显示在页面上
}
```

此段代码是输出 1~100 之间含有因子 3 的数。

**注意：**如果 `continue` 语句陷于多重循环结构之中，则它只对包含它的最内层循环有效。

## 2. break语句

语法形式：

```
break;
```

`break` 语句只能用于循环语句或 `switch` 语句中。如果在 `switch` 语句中执行到 `break` 语句，则立刻从 `switch` 语句中跳出，转到 `switch` 语句的下一条语句；如果在循环语句中执行到 `break` 语句，则循环立刻结束，跳转到循环语句的下一条语句。不管循环有多少层，`break` 语句只能从包含它的最内层循环跳出一层。例如：

```
int m=0;
string mystring = "laskdjflasdkjasdalfakeoflkdsa";
foreach(char mychar in mystring)
{
    m++;
    if (mychar == 'a')           //判断迭代变量 mychar 是否为 a 字符
        break;                 //若 mychar 为 a 字符则跳出循环
}
Response.Write("字符串中第 1 个 a 在"+m+"位置"); //页面上输出：字符串中第 1 个 a 在
                                                "2"位置
```

此段代码是查找出字符串中第 1 个 a 所在的位置。

## 3. return语句

语法形式：

```
return;
```

或

return 表达式;

return 语句出现在一个方法内。在方法中执行到 return 语句时，程序流程转到调用这个方法处。如果方法没有返回值（返回类型修饰为 void），则使用“return”格式返回；如果方法有返回值，那么使用“return 表达式”格式，其中的表达式就是方法的返回值。例如：

```
protected void Page_Load(object sender, EventArgs e)
{
    return; //返回
    Response.Write("程序执行到此"); //页面上没有显示“程序执行到此”字样
}
```

当程序执行到 return 语句时返回空值，并不执行下面语句，所以此段代码并没有在页面上输出“程序执行到此”字样。

#### 4. goto语句

goto 语句可以将程序的执行流程从一个地方转移到另一个地方，非常灵活。但正因为它太灵活，容易造成程序结构混乱的局面，所以应该有节制地、合理地使用 goto 语句。

语法形式：

```
goto 标号;
标号: 语句;
```

其中，“标号”就是定位在某一语句之前的一个标识符，称为标号语句。例如：

```
int m=0;
string mystring = "laskdjflasdkjasdalfakeoflkdsa";
foreach(char mychar in mystring)
{
    m++;
    if(mychar == 'a') //判断迭代变量 mychar 是否为 a 字符
        goto end; //若 mychar 为 a 字符则跳出循环
}
end: Response.Write("第 1 个 a 在第"+m+"位"); //页面上输出：第 1 个 a 在第 2 位
```

上述代码给出了 goto 语句转向的目标，在页面上输出“第 1 个 a 在第 2 位”字样。值得注意的是，goto 语句不能使控制转移到另一个语句块内部，更不能转移另一个函数内部。另外，goto 语句用在 switch 语句中时有如下形式。

语法形式：

```
goto case 常量;
goto default ;
```

它只能在本 switch 语句中从一种情况转向另一种情况。

**【例 2-11】** 在网站 Chapter2 上添加一个网页，命名为 example2-11，利用 goto 语句查询课程表。

设计步骤如下：

- (1) 在 Chapter2 网站上添加一个命名为 “example2-11” 的网页。
- (2) 从工具栏中拖曳 1 个 DropDownList (下拉框) 控件、1 个 Button 控件和 1 个 Lable 控件到页面上。在 Button 控件所对应的属性窗口中, 将 “Text” 属性值 “Button1” 修改为 “查询”。
- (3) 在 DropDownList 控件所对应的属性窗口中选择 “Items” 属性 (列表中项的集合), 单击后面的按钮, 如图 2.21 所示。系统弹出 “ListItem 集合编辑器” 对话框, 添加星期一到星期日, 并且星期日的 “Selected” 属性选择 “True”, 其他的 “Selected” 属性选择 “False”, 如图 2.22 所示。

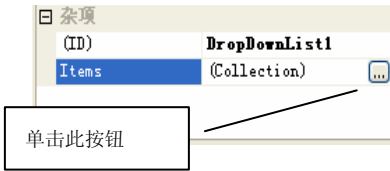


图 2.21 选择 Items 属性



图 2.22 “ListItem 集合编辑器” 对话框

- (4) 双击 Button 控件, 系统自动切换到 “example2-11.aspx.cs” 代码页, 并且添加了 “Button1\_Click” 方法, 在 “Button1\_Click” 方法中添加如下代码:

```
string week = DropDownList1.Text;           //单击下拉框所选择的星期
switch(week)
{
    case "星期六":
    case "星期日":
        Label1.Text = "今天是周末, 自行安排! ";
        break;
    case "星期一":
        goto case "星期四";                 //星期一和星期四的课程一样, 跳到星期四
    case "星期二":
        Label1.Text = "今天的课程是: 数学、英语、体育";
        break;
    case "星期三":
        Label1.Text = "今天下午政治学习";
    case "星期四":
        Label1.Text = "今天的课程是: 数学、英语、C#";
        break;
    case "星期五":
        Label1.Text = "今天下午打扫卫生";
    default: Label1.Text = "数据有错, 没有此选项! ";
        break;
}
```

- (5) 按 【Ctrl+F5】 组合键运行网页, 选择 “星期一”, 单击 【查询】 按钮, 结果如图 2.23 所示。

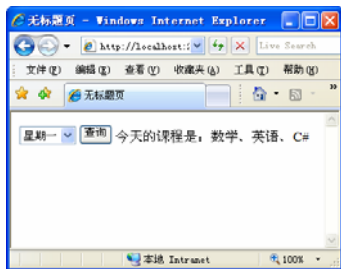


图 2.23 example2-11 网页运行的结果

## 2.5.4 异常处理

程序中对异常的处理使程序更加健壮。现在的许多程序设计语言都增加了异常处理的能力，C#也不例外。异常产生的原因主要有 2 点。

(1) 由 `throw` 语句立即无条件地引发异常，控制永远不会到达紧跟在 `throw` 语句后的语句。

(2) 在处理 C#语句和表达式的过程中，会出现一些例外情况，使某些操作无法正常完成，此时就会引发一个异常。例如，在整数除法运算中，如果分母为零，就会引发一个 `DivideByZeroException` 异常。

异常处理语法形式：

```
try
{
    语句
}catch(类型 标示符)
{
    语句
}finally
{ 语句 }
```

如果执行 `try` 块时出现异常则转向到相应的 `catch` 块，执行完 `catch` 块后再执行 `finally` 块。`finally` 块总是在离开 `try` 语句块后执行的而且 `finally` 块中的程序是必须执行的，`finally` 块主要是释放资源。

**【例 2-12】** 在网站 Chapter2 上添加一个网页，命名为 `example2-12`，利用 `try-catch-finally` 语句处理异常。

设计步骤如下：

(1) 在 Chapter2 网站上添加一个命名为“`example2-12`”的网页。

(2) 在“`example2-12.aspx.cs`”代码页中的“`Page_Load`”方法中添加如下代码：

```
int a = 5, b = 0;
try
{
    a /= b; //不能除以零所以抛出异常
}catch (DivideByZeroException de)
{
```

```

        Response.Write(de.Message);           //页面上输出“试图除以零。”
        return;                               //返回
    }
    finally
    {
        Response.Write("执行到 finally 块中"); //页面上输出“执行到 finally 块中”
    }

```

(3) 按【Ctrl+F5】组合键运行网页，结果如图 2.24 所示。

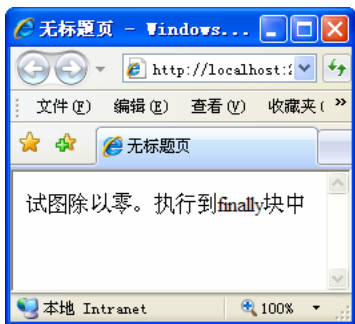


图 2.24 example2-12 网页运行的结果

在 try 块中因为不能除以零所以抛出异常，catch 捕获了此异常，虽然当程序执行到 return 语句要返回，但还是执行了 finally 块。

## 2.6 数组、结构和枚举

### 2.6.1 数组

数组是一种包含若干变量的数据结构，这些变量都具有相同的数据类型并且排列有序，因此可以用一个统一的数组名和下标唯一地确定数组中的元素。

#### 1. 一维数组

一维数组是以线性方式存储固定数目的项的数组。

##### (1) 一维数组的声明

语法形式：

```
type [] arrayName;
```

其中：

- type 可以是 C# 中任意的数据类型；
- [] 表明后面的变量是一个数组类型，必须放在数组名之前；
- arrayName 是数组名，遵循标识符的命名规则。

例如：

```
int [] a1;           // a1 是一个含有 int 类型数据的数组
```

```
double [] f1;           // f1 是一个含有 double 类型数据的数组
string [] s1;           // s1 是一个含有 string 类型数据的数组
```

## (2) 创建数组对象

用 `new` 运算符创建数组实例有两种基本形式：声明数组和创建数组。它们分别进行。  
语法形式：

```
type [] arrayName ;           // 声明数组
arrayName = new type [size];  // 创建数组实例
```

`size` 表明数组元素的个数。

声明数组和创建数组实例也可以合在一起写：

```
type [] arrayName = new type [size] ;
```

例如：

```
int [] a1;
a1 = new int [10];           // a1 是一个有 10 个 int 类型元素的数组
string [] s1 = new string [5]; // s1 是含有 5 个 string 类型元素的数组
```

## (3) 一维数组的初始化

### ● 语法形式 1：

```
type [] arrayName = new type [size] { val1, val2, ...,valn};
```

数组声明与初始化同时进行，`size` 就是数组元素的个数。它必须是常量，而且应该与大括号内的数据个数一致。

### ● 语法形式 2：

```
type [] arrayName = new type [] { val1, val2, ...,valn };
```

默认 `size`，由编译系统根据初始化表中的数据个数，自动计算数组的大小。

### ● 语法形式 3：

```
type [] arrayName = { val1, val2, ...,valn };
```

数组声明与初始化同时进行，还可以默认 `new` 运算符。

### ● 语法形式 4：

```
type [] arrayName ;
arrayName = new type [size] { val1, val2,...,valn };
```

把数组声明与初始化分开在不同的语句中进行时，`size` 同样可以默认，也可以是一个变量。

例如：

```
int [] nums = new int [10] {0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }; //使用语法形式 1 初始化
int [] nums = new int [] {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};   //使用语法形式 2 初始化
int [] nums = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };             //使用语法形式 3 初始化
```

```
int [] nums ;  
nums = new int [10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };           //使用语法形式 4 初始化
```

以上数组初始化实例都是等同的。

#### (4) 一维数组的访问

数组具有初值时，就可以像其他变量一样被访问，既可以取数组元素的值，又可以修改数组元素的值。在 C#中是通过数组名和数组元素的下标来引用数组元素的。

语法形式：

数组名[下标]

下标是数组元素的索引值，实际上就是要访问的那个数组元素在内存中的相对位移。

注意：相对位移是从 0 开始的，所以下标的值从 0 到数组元素的个数-1 为止。

**【例 2-13】** 在网站 Chapter2 上添加一个网页，命名为 example2-13，定义一个一维数组并查找出数组中的最大值和最小值。

设计步骤如下：

- (1) 在 Chapter2 网站上添加一个命名为“example2-13”的网页。
- (2) 在“example2-13.aspx.cs”代码页中的“Page\_Load”方法中添加如下代码：

```
int max, min;  
int[] queue = new int[10] { 89, 78, 65, 52, 90, 92, 73, 85, 91, 95 }; //定义一个一维数组  
max = min = queue[0];           //queue[0]是数组第一个数  
for (int i = 1; i < 10; i++)  
{  
    if (queue[i] > max) max = queue[i];  
    if (queue[i] < min) min = queue[i];  
}  
Response.Write("最大数是:"+max+"<br>最小数是:"+ min);
```

- (3) 按【Ctrl+F5】组合键运行网页，结果如图 2.25 所示。



图 2.25 example2-13 网页运行的结果

## 2. 多维数组

多维数组指能用多个下标访问的数组，多维数组类似于同类型值的矩阵。

#### (1) 多维数组的声明

语法形式：



```
type [ , , , ] arrayName ;
```

在声明时方括号内加逗号，就表明是多维数组，有  $n$  个逗号，就是  $n+1$  维数组。例如：

```
int [ , ] score;           // score 是一个 int 类型的二维数组
float [ , , ] table;       // table 是一个 float 类型的三维数组
```

## (2) 创建数组对象

声明数组和创建数组分别进行。

语法形式：

```
type [ , , , ] arrayName ;           // 声明数组
arrayName = new type [size1, size2, size3]; // 创建数组实例
```

size1、size2、size3 分别表明多维数组每一维的元素个数。

声明数组和创建数组实例也可以合在一起写。

语法形式：

```
type [ , , , ] arrayName = new type [size1, size2, size3] ;
```

例如：

```
int [ , ] score ;
score = new int [3, 4] ;           // score 是一个 3 行 4 列的二维数组
float [ , , ] table=new float [2, 3, 4] // table 是一个三维数组，每一维的维数分别是 2、3、4
```

## (3) 多维数组的初始化

多维数组的初始化是将每维数组元素设置的初始值放在各自的大花括号内，下面以最常用的二维数组为例来讨论。

### ● 语法形式 1：

```
type [ , ] arrayName = new type [size1, size2 ] {{ val11, val12, ...,val1n },
{ val21, val22, ...,val2n }, ..., { valm1, valm2, ...,valmn }};
```

数组声明与初始化同时进行，数组元素的个数是  $\text{size1}*\text{size2}$ ，数组的每一行分别用一个花括号括起来，每个花括号内的数据就是这一行的每一列元素的值，初始化时的赋值顺序按矩阵的“行”存储原则。

### ● 语法形式 2：

```
type [ ] arrayName = new type [ , ] {{ val11, val12, ...,val1n },
{ val21, val22,...,val2n }, ..., { valm1, valm2, ...,valmn }};
```

默认 size，由编译系统根据初始化表中花括号{}的个数确定行数，再根据{}内的数据确定列数，从而得出数组的大小。

### ● 语法形式 3：

```
type [ , ] arrayName ={{ val11, val12, ...,val1n },
{ val21, val22, ...,val2n }, ..., { valm1, valm2, ...,valmn }};
```

数组声明与初始化同时进行，还可以默认 new 运算符。

● 语法形式 4:

```
type [ , ] arrayName ;
arrayName = new type [size1, size2] {{ val11, val12, ...,val1n },{ val21, val22, ...,val2n }, ...,
{ valm1, valm2, ...,valmn }};
```

把数组声明与初始化分开在不同的语句中时，size1、size2 同样可以默认，但也可以是变量。  
例如：

```
int [ , ] a = new int [3,4] {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}}; //使用语法形式 1 初始化
int [ , ] a = new int [ , ] {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}}; //使用语法形式 2 初始化
int [ , ] a = {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}}; //使用语法形式 3 初始化
int [ ] a ;
a = new int [3, 4] {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}}; //使用语法形式 4 初始化
```

以上数组初始化实例都是等同的。

**【例 2-14】** 在网站 Chapter2 上添加一个网页，命名为 example2-14，求两个矩阵的乘积。  
设计步骤如下：

- (1) 在 Chapter2 网站上添加一个命名为 “example2-14” 的网页。
- (2) 在 “example2-14.aspx.cs” 代码页中的 “Page\_Load” 方法中添加如下代码：

```
int i,j,k;
int [ , ] a = new int [3,4]{{1,2,3,4},{5,6,7,8}, {9,10,11,12}};
int [ , ] b = new int [4,3]{{12,11,10},{9,8,7},{6,5,4}, {3,2,1}};
int [ , ] c = new int [3,3];
for (i=0; i<3; i++)
for (j=0; j<3; j++)
    for (k=0;k<4;++k)
        c[i, j]+= a[i, k]*b[k, j];
for (i=0; i<3; ++i)
{
    for (j=0; j<3; ++j)
        Response.Write(c[i,j]+" ");
    Response.Write("<br>");
}
```

- (3) 按 **【Ctrl+F5】** 组合键运行网页，结果如图 2.26 所示。



图 2.26 example2-14 网页运行的结果

## 2.6.2 结构

结构是值类型，通过结构可以把相关的多种类型数据（变量）组成整体来处理，组成结构类型的各个变量称为结构的数据成员（简称成员，或者成员变量）。所有的基类型都是结构类型，例如，`int` 对应 `System.int32` 结构，`string` 对应 `System.string` 结构。通过使用结构可以创建更多的值类型。

语法形式：

```
[访问修饰符] struct [结构类型名]
{
    [访问级别]<成员定义 1>
    [访问级别]<成员定义 2>;
    [访问级别]<成员定义 3>;
    ...
}
```

其中访问修饰符包括 `public`（公共的）、`protected`（保护的）、`private`（私有的）。它们的访问级别是由高到低的。

例如：

```
public struct Student
{
    public int number;           //学生的学号
    public string name;         //学生的姓名
    public int age;              //学生的年龄
    public string sex            //学生的性别
}
```

上述代码声明一个学生结构。

**【例 2-15】** 在网站 Chapter2 上添加一个网页，命名为 `example2-15`，利用结构在页面上输出学生的信息。

设计步骤如下：

- （1）在 Chapter2 网站上添加一个命名为“`example2-15`”的网页。
- （2）在“`example2-15.aspx.cs`”代码页中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    Student stu;           //声明一个学生类型对象 stu
    stu.age = 21;           //初始化 stu 的 age
    stu.name = "王明";      //初始化 stu 的 name
    stu.number = 81101;     //初始化 stu 的 number
    stu.sex = "男";         //初始化 stu 的 sex
    Response.Write("姓名: "+stu.name+"<br>学号: "+stu.number+"<br>性别: "+
        stu.sex+"<br>年龄: "+stu.age); //在页面上输出学生的信息
}
```

```

    }
    protected struct Student
    {
        public int number;
        public string name;
        public int age;
        public string sex;
    }
    //定义学生结构类型

```

(3) 按【Ctrl+F5】组合键运行网页，结果如图 2.27 所示。

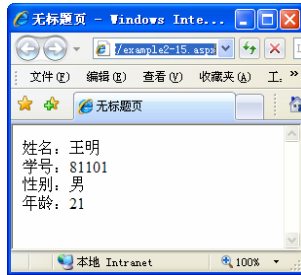


图 2.27 example2-15 网页运行的结果

### 2.6.3 枚举

枚举应用于有多个选择情况的场合，枚举类型为一组符号常数提供了一个类型名称。枚举中的每个成员实际上是一个符号常数。例如：

```

enum Color
{
    Red,
    Green,
    Blue
}

```

它声明了一个枚举类型 **Color**，表示三种可能的情况：**Red**、**Green** 和 **Blue**。这里的三个值实际上是三个整数 0、1、2，但与整数相比，使用枚举类型使程序的可读性更好，并且容易检查出错误。声明枚举类型，使用关键字 **enum**。声明的基本语法格式如下：

```

enum 枚举名[: 基本类型名]
{
    枚举成员 [= 常数表达式],
    .....
}

```

每个枚举类型都有一个相应的整数类型，称为枚举类型的基本类型。一个枚举类型声明可以显式地声明 **byte**、**sbyte**、**short**、**ushort**、**int**、**uint**、**long** 或 **ulong** 中的一个基本类型。

如果没有显式地声明基本类型，则默认为 **int**。枚举类型声明的主体定义了零个至多个枚举成员，它们是枚举类型命名的常数。

一个枚举成员的数值，既可以使用等号“=”显式地赋值，也可以不显式地赋值，而使用隐式赋值。隐式赋值按以下规则来确定值。

(1) 对第一个枚举成员，如果没有显式赋值，则其数值为 0。

(2) 对其他枚举成员，如果没有显式赋值，则其值等于前一枚举成员的值加 1。

例如：

```
enum Color
{
    Red,
    Green = 10,
    Blue,
    Max = Blue
}
```

其中，Red 的值为 0，Green 的值为 10，Blue 的值为 11，Max 的值为 11。

枚举成员前面不能显式地使用修饰符。每个枚举成员隐含都是 const 的，其值不能改变；每个成员隐含都是 public 的，其访问控制不受限制；每个成员隐含都是 static（静态）的，直接用枚举类型名进行访问。

枚举类型的 ToString() 方法能得到一个字符串，这个字符串是相对应的枚举成员的名称。

**【例 2-16】** 设计一个网页，将阿拉伯数字转换为大写的汉字。

设计步骤如下所示：

(1) 在 Chapter2 网站上添加一个命名为“example2-16”的网页。

(2) 从工具栏中拖拽 1 个 TextBox 控件、1 个 Button 控件和 1 个 Label 控件到页面上。在 Button 控件所对应的属性窗口中，将“Text”属性值“Button1”修改为“转换”。

(3) 双击 Button 控件，添加代码，代码如下所示。

```
protected void Button1_Click(object sender, EventArgs e)
{
    string sin = TextBox1.Text.Trim();           //存放输入的数字
    string sout=null;                             //存放转换后的大写汉字
    foreach (char s in sin)
    {
        int a = Convert.ToInt32(s) - 48;         //字符'0'的 ASCII 码的十进制是 48
        switch (a)
        {
            case (int)ChineseChar.零:             //显式类型转换
                sout += ChineseChar.零.ToString(); //获得枚举成员的名字
                break;
            case (int)ChineseChar.壹:
                sout += ChineseChar.壹.ToString();
                break;
            case (int)ChineseChar.贰:
                sout += ChineseChar.贰.ToString();
                break;
            case (int)ChineseChar.叁:
```

```

        sout+=ChineseChar.叁.ToString();
        break;
    case (int)ChineseChar.肆:
        sout+=ChineseChar.肆.ToString();
        break;
    case (int)ChineseChar.伍:
        sout+=ChineseChar.伍.ToString();
        break;
    case (int)ChineseChar.陆:
        sout+=ChineseChar.陆.ToString();
        break;
    case (int)ChineseChar.柒:
        sout+=ChineseChar.柒.ToString();
        break;
    case (int)ChineseChar.捌:
        sout+=ChineseChar.捌.ToString();
        break;
    case (int)ChineseChar.玖:
        sout+=ChineseChar.玖.ToString();
        break;
    }
    Label1.Text = sout;
}

enum ChineseChar : int                                //声明一个 int 基本类型的数据大写枚举
{
    零,壹,贰,叁,肆,伍,陆,柒,捌,玖
}

```

(4) 按【Ctrl+F5】组合键运行网页，输入数据“123456789987654321”，单击【转换】按钮，结果如图 2.28 所示。

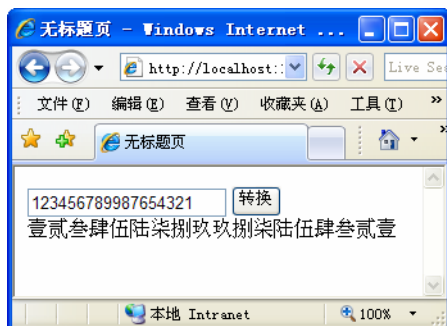


图 2.28 example2-16 网页运行的结果

## 习 题

1. C#中的数据类型可以分为值类型和\_\_\_\_\_。

2. 下列类型属于引用类型的有 ( )。
- A. 类类型                      B. 结构体                      C. 数组                      D. 枚举
3. 装箱转换是指将一个值类型隐式或显式地转换成一个\_\_\_\_\_类型。
4. 装箱操作可以隐式进行, 但拆箱操作必须是\_\_\_\_\_的。
5. 可以终止并跳出循环的语句是 ( )。
- A. break 语句      B. continue 语句      C. goto 语句      D. return 语句
6. 下列运算符是三元运算符的是 ( )。
- A. &&                      B. +                      C. ? :                      D. --
7. 在 C#中, 下列哪些语句可以创建一个具有 3 个初始值为 ABC 的元素的字符串数组 ( )。
- A.string strlist[3](ABC)
- B. string[3]strlist={ABC,ABC,ABC}
- C. string[] strlist = { "ABC","ABC","ABC"}
- D. string[]strlist=new string[3]
8. 下列说法正确的有 ( )
- A. switch 选择结构中必须有 default 标记。
- B. switch 选择结构的每个 case 中必须有 break 语句才能正确退出这个 case 结构。
- C. 如果表达式 (X>Y) 为真或 (A<B) 为真, 则表达式 (X>Y&&A<B) 为真。
- D. 如果至少有一个操作数为真, 则包括||运算符的表达式为真。
9. 在 C#中, 将路径名 “D:\Program Files\DB2” 存入字符串变量 path 中的的语句有\_\_\_\_\_和\_\_\_\_\_。
10. C#中循环语句包括\_\_\_\_\_、\_\_\_\_\_和\_\_\_\_\_。
11. 在 C#程序中, 可以使用 try..catch 机制来处理程序出现的 ( ) 错误。
- A. 语法                      B. 运行                      C. 逻辑                      D. 拼写
12. 在页面上打印九九乘法表, 打印结果如图 2.29 所示。



图 2.29 九九乘法表

13. 在页面上打印出所有的“水仙花数”。所谓“水仙花数”是指一个 3 位数, 各个位数的立方和等于该数本身。例如, 153 是一个“水仙花数”, 因为  $153=1^3+5^3+3^3$ 。
- 提示: 利用 for 循环控制 100~999, 每个数分解出个位、十位和百位。
14. 利用 while 语句验证“角谷猜想”。所谓“角谷猜想”是指将一个自然数按以下的一

个简单规则进行运算：若为偶数，则除以 2；若为奇数，则乘 3 并加 1。将得到的数重复再按该规则运算，最终可得到 1。

15. 定义一个数组并给数组赋值，利用 `foreach` 语句在页面上显示出数组中的数据。

16. 显示字符的 ASCII 码。拖放一个 `TextBox` 控件到页面用于接收输入的字符，再拖放一个 `Button` 控件到页面用于执行转换并显示在页面上。

17. 编程求 1000 之内的所有“完数”并显示在页面上。所谓“完数”是指一个数恰好等于它的因子之和。例如，6 是“完数”，因为  $6=1+2+3$ 。

18. 编写程序，求 100~200 之间的所有质数并显示在页面上。



## 第 3 章 C#面向对象编程

早期的程序开发使用过程化的设计方法，对于大型应用程序的开发显得力不从心，后续的维护也比较困难。而面向对象的编程方式把客观世界中的业务及操作对象转变为计算机中的对象，这样，程序开发者能够以更趋近于人的思维方式来编程。这使得程序更易理解，开发效率大大提高，维护也更容易。下面介绍面向对象编程的基本知识。

### 3.1 类和对象

现实世界是由各种各样的实体对象所组成的。每种对象都有自己的内部状态和运动规律，不同对象之间的相互联系和相互作用构成了各种不同的系统，并进而构成整个客观世界。同时人们为了更好地认识客观世界，把具有相似内部状态和运动规律的实体对象综合在一起称为类。类是具有相似内部状态和运动规律的实体的抽象，进而人们抽象地认为客观世界是由不同类的事物相互联系和相互作用所构成的一个整体。

#### 3.1.1 创建类和对象

对象是面向对象语言的核心，数据抽象和对象封装是面向对象技术的基本要求，而实现这一切的主要手段和工具就是类。从编程语言的角度讲，类就是一种数据结构，它定义数据和操作这些数据的代码。类是对象的数据抽象，实例化后的类为对象。

##### 1. 类的声明

要定义一个新的类，首先要声明它。

语法形式：

```
[属性集信息] [类修饰符] class 类名 [: 类基]
{
    [类成员]
}
```

其中：

- 属性集信息：C#语言提供给程序员的，为程序中定义的各种实体附加一些说明信息。这是 C#语言的一个重要特征。
- 类修饰符：可以是表 3.1 所列的几种之一或是它们的有效组合，但在类声明中，同一修饰符不允许出现多次。
- 类基：它定义该类的直接基类和由该类实现的接口；当多于一项时，用逗号“,”分隔。如果没有显式地指定直接基类，那么它的基类隐含为 object。

表 3.1 类修饰符

修 饰 符	作 用 说 明
public	表示不限制对类的访问。类的访问权限省略时默认为 public
protected	表示该类只能被这个类的成员或派生类成员访问
private	表示该类只能被这个类的成员访问
internal	表示该类能够由程序集中的所有文件使用，而不能由程序集之外的对象使用
new	只允许用在嵌套类中，它表示所修饰的类会隐藏继承下来的同名成员
abstract	表示这是一个抽象类，该类含有抽象成员，因此不能被实例化，只能用做基类
sealed	表示这是一个密封类，不能从这个类再派生出其他类。显然密封类不能同时为抽象类

例如：

```
class Point                // Point 类的访问权限默认为 public
{
    int x, y;              //类的成员
}
```

2. 类的成员

类的定义包括类头和类体两部分，其中类体用一对大花括号 { }括起来，类体用于定义该类的成员。

语法形式：

```
{
    [类成员声明]
}
```

类的成员由两部分组成，一个是以类成员声明形式引入的类成员，另一个则是直接从它的基类继承而来的成员。类成员声明主要包括常数声明、字段声明、方法声明、属性声明、事件声明、索引器声明、运算符声明、构造函数声明、析构造函数声明、静态构造函数声明、类型声明等。当字段、方法、属性、事件、运算符和构造函数声明中含有 static 修饰符时，则表明它们是静态成员，否则就是实例成员。

(1) 访问修饰符

类成员声明中可以使用表 3.2 中的 5 种访问修饰符中的一种。当类成员声明不包含访问修饰符时，默认约定访问修饰符为 private。

表 3.2 类成员访问修饰符

修 饰 符	作 用 说 明
public	同一程序集中的任何其他代码或引用该程序集的其他程序集都可以访问该类型或成员
protected	只有同一类或结构或者派生类中的代码可以访问该类型或成员
private	只有同一类或结构中的代码可以访问该类型或成员
internal	同一程序集中的任何代码都可以访问该类型或成员，但其他程序集中的代码不可以
protected	同一程序集中的任何代码或其他程序集中的任何派生类都可以访问该类型或成员

## (2) 常数声明

语法形式:

```
[属性集信息] [常数修饰符] const 类型 标识符 = 常数表达式 [, ...]
```

其中:

- 常数修饰符: new、public、protected、internal、private。
- 类型: sbyte、byte、short、ushort、int、uint、long、ulong、char、float、double、decimal、bool、string、枚举类型或引用类型。常数表达式的值类型应与目标类型一致, 或者通过隐式转换规则转换成目标类型。

例如:

```
class A_const
{
    public const int X=10;           //public 修饰符
    const double PI=3.14159;        //默认访问修饰符, 即约定为 private
    const double Y= 0.618+3.14;
}
```

常数表达式的值应该是一个可以在编译时计算的值。常数声明不允许使用 static 修饰符, 但它和静态成员一样只能通过类访问。

**【例 3-1】** 设计一个网站, 命名为 Chapter3, 添加一个网页, 命名为 example3-1, 用于新建一个类并且访问类中的常量。

设计步骤如下:

① 在 D 盘新建一个命名为 “Chapter3” 的文件夹, 打开 VS 2008, 依次展开工具栏中的 “新建” → “网站” 选项, 系统弹出 “新建网站” 对话框, 选择 “ASP.NET 网站” 模板, 语言选择 “Visual C#”, 框架选择 “.NET Framework 3.5”, 位置选择 “D:\Chapter3”, 单击【确定】按钮。

② 打开 “资源管理器” → 右击 “D:\Chapter3\” 网站 → 单击 “添加新项” 选项, 系统弹出 “添加新项” 对话框, 选择 “Web 窗体” 并且重新命名为 “example3-1”, 单击【添加】按钮。

③ 打开 “资源管理器” → 打开刚添加的 “example3-1.aspx.cs” 文件, 添加如下代码。

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("X="+A_const.X);    //通过 A_const 类访问 X 常量, 页面输出: X=10
}
class A_const                           //A_const 类的访问权限默认为 public
{
    public const int X=10;               //public 修饰符
    const double PI=3.14159;            //默认访问修饰符, 即约定为 private
    const double Y= 0.618+3.14;
}
```

④ 按【Ctrl+F5】组合键运行网页，结果如图 3.1 所示。



图 3.1 example3-1 运行的结果

(3) 字段声明

语法形式：

```
[属性集信息] [字段修饰符] 类型 变量声明列表;
```

其中：

- 变量声明列表：标识符或用逗号“，”分隔的多个标识符，并且变量标识符还可用赋值号“=”设定初始值。例如：

```
class A
{
    int x=100, y = 200;           //声明 int 类型字段
    float sum = 1.0f;           //声明 float 类型字段
}
```

- 字段修饰符：new、public、protected、internal、private、static、readonly、volatile。加 static 修饰的字段是静态字段，不加 static 修饰的字段是实例字段；加 readonly 修饰的字段是只读字段。

3. 创建类的对象

类和对象是紧密结合的，类是对象总体上的定义，而对象是类的具体实现。创建类的对象时需要使用关键字 new。

语法形式：

```
类名 对象名=new 类名();           //类名()是构造函数
```

例如，在【例 3-1】的“page\_load”方法中创建类 A\_const 的一个对象 m：

```
A_const m=new A_const();           //创建了一个 m 对象
```

3.1.2 构造函数和析构函数

1. 构造函数

当定义了一个类之后，就可以通过 new 运算符将其实例化，产生一个对象。为了能规范、

安全地使用这个对象，C#提供了实现对象进行初始化的方法，这就是构造函数。

在 C#中，类的成员字段可以分为实例字段和静态字段，与此相应的构造函数也分为实例构造函数和静态构造函数。

(1) 实例构造函数的声明

语法形式:

```
[属性集信息] [构造函数修饰符] 标识符 ([参数列表])
[: base ([参数列表])]    [: this ([参数列表])]
{
    构造函数语句块
}
```

其中:

- 构造函数修饰符: public、protected、internal、private、extern。一般地，构造函数总是 public 类型的。如果是 private 类型的，则表明类不能被外部类实例化。
- 标识符 ([参数列表]<sub>opt</sub>): 构造函数名，必须与这个类同名，不声明返回类型，并且没有任何返回值。它与返回值类型为 void 的函数不同。构造函数可以没有参数，也可以有一个或多个。例如:

```
class A
{
    int X,Y;                //声明 int 类型字段
    public A(int x)          //带有一个参数的构造函数
    {    X=x; }              //给字段赋值
    public A(int x,int y)    //带有两个参数的构造函数
    {
        X=x;
        Y=y;
    }
}
```

用 new 运算符创建一个类的对象时，类名后的一对圆括号提供初始化列表，这实际上就是提供给构造函数的参数。系统根据这个初始化列表的参数个数、参数类型和参数顺序调用不同的构造函数。

**【例 3-2】** 在网站 Chapter3 上添加一个网页，命名为 example3-2，用于体现构造函数的使用。

设计步骤如下:

- ① 在 Chapter3 网站上添加一个命名为“example3-2”的网页。
- ② 在“example3-2.aspx.cs”代码页中添加如下代码:

```
protected void Page_Load(object sender, EventArgs e)
{
    Student stu_1 = new Student("081101", "王明", "男", 27);    //创建一个对象 stu_1
    Response.Write("学号: "+stu_1.XH);                          //页面上输出学号
    Response.Write("<br>姓名: " + stu_1.XM);
```

```

        Response.Write("<br>性别: " + stu_1.XB);
        Response.Write("<br>年龄: " + stu_1.NL);
    }
class Student
{
    public string XH, XM, XB;
    public int NL;
    public Student(string xh,string xm,string xb,int nl)           //构造函数
    {
        XH = xh;                                                  //对 XH 字段初始化
        XM = xm;
        XB = xb;
        NL = nl;
    }
}

```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.2 所示。



图 3.2 example3-2 网页运行的结果

(2) 静态构造函数的声明  
语法形式:

```

[属性集信息] [静态构造函数修饰符] 标识符()
{
    静态构造函数体
}

```

其中:

- 静态构造函数修饰符: [ extern ] static 或者 static [ extern ]。如果有 extern 修饰，则说明这是一个外部静态构造函数，不提供任何实际的实现，所以静态构造函数体仅仅是一个分号。
- 标识符(): 是静态构造函数名，必须与这个类同名；静态构造函数不能有参数。
- 静态构造函数体: 静态构造函数的目的是对静态字段的初始化，所以它只能对静态数据成员进行初始化，而不能对非静态数据成员进行初始化。

例如：

```
class Student
{
    static string XH, XM, XB;
    static int NL;
    static Student()           //静态构造函数
    {
        // 静态构造函数，对类的静态字段初始化
        XH = "081101";
        XM = "王明";
        XB = "男";
        NL = 21;
    }
}
```

## 2. 析构函数

一般来说，创建一个对象时需要用构造函数初始化数据。与此相对应，释放一个对象时就用析构函数。所以析构函数是用于实现析构类实例所需操作的方法。

语法形式：

```
[属性集信息] [ extern ] ~标识符 ()
{
    析构函数体
}
```

其中：

- 标识符：必须与类名相同，但为了区别于构造函数，前面需加“~”，表明它是析构函数。
- 析构函数：不能写返回类型，也不能带参数，一个类最多只能有一个析构函数。

例如：

```
class A
{
    int X;
    public A()           //不带参数的构造函数
    {    X=10;    }
    ~A()                //析构函数
    {    }
}
```

如果没有显式地声明析构函数，则编译器将自动产生一个默认的析构函数。

**注意：**析构函数不能由程序显式地调用，而是由系统在释放对象时自动调用的。

## 3.2 方法

C#实现了完全意义上的面向对象，它没有全局常数、全局变量和全局方法，任何事物都必须封装在类中。通常，程序的其他部分通过类所提供的方法与它进行互操作。对方法的理解可以从方法的声明、方法的参数、静态方法与实例方法、方法的重载与覆盖等方面切入。

### 3.2.1 方法的声明

方法是按照一定格式组织的一段程序代码，在类中用方法声明的方式来定义。  
语法形式：

```
[属性集信息] [方法修饰符] 返回类型 方法名 ( [形参表] )
{
    方法体
}
```

其中：

- 方法修饰符：如表 3.3 所示。方法修饰符中的 `public`、`protected`、`private`、`internal`、`protected internal` 属于访问修饰符，表示访问的级别，默认情况下，方法的访问级别为 `public`。

表 3.3 方法修饰符

修 饰 符	作 用 说 明
new	在一个继承结构中，用于隐藏基类同名的方法
public	表示该方法可以在任何地方被访问
protected	表示该方法可以在其类体或派生类类体中被访问，但不能在类体外访问
private	表示该方法只能在这个类体内被访问
internal	表示该方法可以被同处于一个工程的文件访问
static	表示该方法属于类型本身，而不属于某特定对象
virtual	表示该方法可在派生类中重写，来更改该方法的实现
abstract	表示该方法仅仅定义了方法名及执行方式，但没有给出具体实现，所以包含这种方法的类是抽象类，有待于派生类的实现
override	表示该方法是将基类继承的 <code>virtual</code> 方法的新实现
sealed	表示这是一个密封方法，它必须同时包含 <code>override</code> 修饰，以防止其派生类进一步重写该方法
extern	表示该方法从外部实现

- 返回类型：方法可以返回值也可以不返回值。如果返回值，则需要说明返回值的类型，它可以是任何一种 C#的数据类型，在方法体内通过 `return` 语句将数据交给调用者。如果不返回值，则其返回类型可标为 `void`，默认情况下为 `void`。
- 方法名：每个方法都有一个名称，一般可以按标识符的起名规则随意给定方法名。不过要记住 `Main()`是为开始执行程序的方法预留的，另外不要使用 C#的关键字作为方法名。为了使方法容易理解和记忆，建议方法的命名尽可能地与所要进行的操作联系起来，即通常说的顾名思义。



■ 形参表：由零个或多个用逗号分隔的形式参数组成，形式参数可用属性、参数修饰符、类型等描述。当形参表为空时，外面的圆括号也不能省略。

■ 方法体：用花括号括起的一个语句块。

例如，有三门课程，分别是语文（YW）、英语（YY）、数学（SX），可以在学生类中添加计算学生的这三门课程平均成绩的方法，如下所示：

```
class Student
{
    ...                                //构造函数等在这里省略
    public int averageScore(int YW,int SX,int YY) //声明计算平均成绩的方法
    {
        int a = (YW + SX + YY)/3;        //计算平均成绩并赋值给变量 a
        return a;                        //返回 a 值
    }
}
```

### 3.2.2 方法的参数

参数的功效就是能使信息在方法中传入或传出。当声明一个方法时，包含的参数说明是形式参数（形参）。当调用一个方法时，给出的对应实际参数是实在参数（实参）。参数的传入或传出就是在实参与形参之间发生的，在 C# 中实参与形参有 4 种传递方式。

#### 1. 值参数

在方法声明时不加修饰的形参就是值参数，它表明实参与形参之间按值传递。当这个方法被调用时，编译器为值参数分配存储单元，然后将对应的实参的值复制到形参中。实参可以是变量、常量、表达式，但要求其值的类型必须与形参声明的类型相同或能够被隐式地转化为这种类型。这种传递方式的好处是，在方法中对形参的修改不影响外部的实参，也就是说，数据只能传入方法而不能从方法传出，所以值参数有时也被称为入参数。

#### 2. 引用参数

如果调用一个方法，期望能够对传递给它的实际变量进行操作，则用 C# 默认的按值传递是不可能实现的。所以 C# 用 **ref** 修饰符来解决此类问题，它告诉编译器，实参与形参的传递方式是引用参数。

引用参数与值参数不同，引用参数并不创建新的存储单元，它与方法调用中的实参变量同处一个存储单元。因此，在方法内对形参的修改就是对外部实参变量的修改。

#### 3. 输出参数

在参数前加 **out** 修饰符的被称为输出参数，它与 **ref** 参数相似，只有一点除外，就是输出参数只能用于从方法中传值，而不能从方法调用处接收实参数据。在方法内 **out** 参数被认为是未赋过值的，所以在方法结束之前应该对 **out** 参数赋值。

## 4. 参数数组

一般而言，调用方法时其实参必须与该方法声明的形参在类型和数量上相匹配，但有时候我们希望更灵活一些，能够给方法传递任意个数的参数。例如，在 3 个数中找最大、最小和在 5 个数中找最大、最小，甚至在任意多个数中找最大、最小能使用同一个方法。C#提供了传递可变长度参数表的机制，使用 `params` 关键字来指定一个可变长度的参数表。

**【例 3-3】** 在网站 Chapter3 上添加一个网页，命名为 `example3-3`，用于体现不同参数方法的使用。

设计步骤如下：

- ① 在 Chapter3 网站上添加一个命名为“`example3-3`”的网页。
- ② 在“`example3.aspx.cs`”代码页中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    int Max;
    Sort S = new Sort();
    Max= S.Sort_1(8,32,12);
    Response.Write("值参数方法，最大值： " + Max);
    int a,b,c;
    a = 2;
    b = 33;
    c = 10;
    S.Sort_2(ref a,ref b,ref c);
    Response.Write("<br>引用参数方法，最大值： " + c);
    S.Sort_3(3, 54, 22,out Max);
    Response.Write("<br>输出参数方法，最大值:" + Max);
    int[] score = { 87, 89, 56, 90, 100, 75, 64, 45, 80, 84 };
    S.Sort_4(out Max,score);
    Response.Write("<br>参数数组方法，最大值:" + Max);
}

class Sort //声明一个类 Sort
{
    public int Sort_1(int x, int y, int z) //值参数方法
    {
        int tmp; // tmp 是方法 Sort_1 的局部变量
        // 将 x, y, z 按从小到大排序
        if (x > y) { tmp = x; x = y; y = tmp; }
        if (x > z) { tmp = x; x = z; z = tmp; }
        if (y > z) { tmp = y; y = z; z = tmp; }
        return z; //返回最大值 z
    }
    public void Sort_2(ref int x, ref int y, ref int z) //引用参数方法
    {
        int tmp; // tmp 是方法 Sort_2 的局部变量
        // 将 x, y, z 按从小到大排序
```

```

        if (x > y) { tmp = x; x = y; y = tmp; }
        if (x > z) { tmp = x; x = z; z = tmp; }
        if (y > z) { tmp = y; y = z; z = tmp; }
    }
    public void Sort_3(int x,int y,int z, out int max)    //输出参数方法
    {
        int tmp;    // tmp 是方法 Sort_3 的局部变量
        // 将 x, y, z 按从小到大排序
        if (x > y) { tmp = x; x = y; y = tmp; }
        if (x > z) { tmp = x; x = z; z = tmp; }
        if (y > z) { tmp = y; y = z; z = tmp; }
        max = z;    //最大值 z 赋给参数 max
    }
    public void Sort_4 (out int max,params int [ ] a )    //参数数组方法
    {
        max = a[0] ;
        for (int i=1; i<a.Length ; i++)
        {
            if (a[i]>max) max=a[i];
        }
    }
}

```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.3 所示。

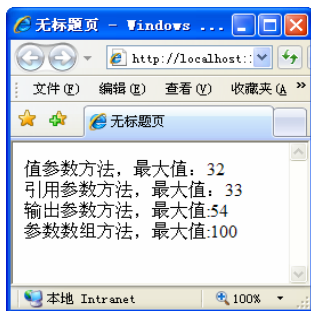


图 3.3 example3-3 网页运行的结果

### 3.2.3 静态方法与实例方法

类的数据成员可以分静态字段和实例字段。静态字段是与类相关联的，不依赖特定对象的存在；实例字段是与对象相关联的，访问实例字段依赖于实例的存在。因此，根据静态字段和实例字段的特性，构造函数将其分为静态构造函数和实例构造函数，方法也将其分为静态方法和实例方法。

通常，若一个方法声明中含有 `static` 修饰符，则表明这个方法是静态方法，同时说明它只对这个类中的静态成员操作，不可以直接访问实例字段。

**【例 3-4】** 下面是一个商品销售管理的简单程序。每一种商品对象要存储的是商品总数及商品单价。每销售一件商品要计算销售额和库存。`cashRegister` 类将销售总额定义为静态变量 `cashSum`，那么访问 `cashSum` 的方法 `productCost` 也就定义为静态方法，而 `makeSale` 方法是计算销售额及库存的，所以定义为实例方法。

设计步骤如下：

- ① 在 Chapter3 网站上添加一个命名为 “example3-4” 的网页。
- ② 切换到设计视图，从工具箱中拖放 4 个 Label 控件、3 个 TextBox 控件和 1 个 Button 控件，如图 3.4 所示。
- ③ 双击 Button 控件，系统自动添加了 “Button\_Click” 方法。

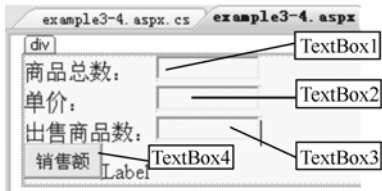


图 3.4 example3-4 页面的设计

- ④ 在 “example3-4.aspx.cs” 代码页中添加如下代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
    int st; //库存量
    int InNumItems = Convert.ToInt32(TextBox1.Text); //所输入的商品总数
    int Incost = Convert.ToInt32(TextBox2.Text); //所输入的商品单价
    int Insale = Convert.ToInt32(TextBox3.Text); //所输入的商品销售数
    cashRegister C = new cashRegister(InNumItems, Incost);
    C.makeSale(Insale,out st); //调用实例方法
    Label4.Text = "销售额: "+cashRegister.productCost().ToString();
    //调用静态方法并转换为 string 类型

    Label4.Text += " 库存量: "+st.ToString(); //库存量转换为 string 类型
}

class cashRegister
{
    int numItems; //商品总数
    double cost; //商品单价
    static double cashSum; //cashSum 是静态变量，计算销售总额
    public cashRegister(int numItems, double cost) //实例构造函数
    {
        this.numItems = numItems;
        this.cost = cost;
    }
    static cashRegister() //静态构造函数
    {
        cashSum = 0.0;
        // this.cashSum=0.0; 错误，静态方法不允许使用 this
    }
    public void makeSale(int num) // 实例方法
    {
        this.numItems -= num;
        cashSum += cost * num; // 实例方法可以访问静态成员
    }
}
```

```

    }
    public static double productCost()                // 静态方法，只能访问静态成员
    {
        return cashSum;
        // return this.cashSum; 错误，静态方法不能使用 this
    }
}

```

⑤ 按【Ctrl+F5】组合键运行网页，单击按钮，结果如图 3.5 所示。

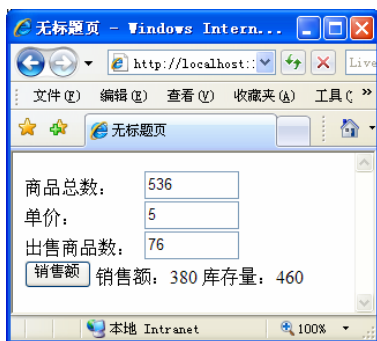


图 3.5 example3-4 网页运行的结果

从上面的程序中可以看到，实例方法 `makeSale()` 中可以使用 `this` 来引用变量 `numItems`。这里，`this` 关键字表示引用当前对象实例的成员。在实例方法体内也可以省略 `this`，直接引用 `numItems`，实际上两者的语义相同。而静态方法不与对象关联，理所当然地不能用 `this` 指针。

### 3.2.4 方法的重载

一个方法的名字和形参的个数、修饰符及类型共同构成了这个方法的签名，同一个类中不能有相同签名的方法。如果一个类中有两个或两个以上的方法同名，而它们的形参个数或形参类型有所不同是允许的，它们属于不同的方法签名。但仅仅是返回类型不同的同名方法，编译器是不能识别的。下面通过一个例子来介绍方法的重载。

**【例 3-5】** 在网站 Chapter3 上添加一个网页，命名为 `example3-5`，用于体现方法的重载使用。

设计步骤如下：

- ① 在 Chapter3 网站上添加一个命名为“`example3-5`”的网页。
- ② 在“`example3-5.aspx.cs`”代码页中添加如下代码：

```

protected void Page_Load(object sender, EventArgs e)
{
    Myclass M = new Myclass();                //实例化一个 Myclass 的对象 M
    int a=10, b=20, c=30;
    double f=3.5, g=5.5, h=1.5;
    Response.Write("a = 10; b = 20; c = 30; f = 3.5; g = 5.5; h = 1.5;");
    Response.Write("<br>a、b 的最大值:" + M.max(a,b));        //调用两个 int 类型参数方法
    Response.Write("<br>a、b、c 的最大值:" + M.max(a, b,c)); //调用三个 int 类型参数方法
    Response.Write("<br>h、f 的最大值:" + M.max(h,f));        //调用两个 double 类型参数方法
}

```

```

        Response.Write("<br>f、g、h 的最大值:" + M.max(f, g, h)); //调用 3 个 double 类型参数方法
    }
    //该类中有 max 方法的 4 个不同版本，它们或者参数类型不同，或者参数个数不同
    class Myclass
    {
        public int max(int x, int y)                //两个 int 类型参数方法
        {
            return x >= y ? x : y;
        }
        public double max(double x, double y)        //两个 double 类型参数方法
        {
            return x >= y ? x : y;
        }
        public int max(int x, int y, int z)          //三个 int 类型参数方法
        {
            return max(max(x, y), z);
        }
        public double max(double x, double y, double z) //三个 double 类型参数方法
        {
            return max(max(x, y), z);
        }
    }

```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.6 所示。

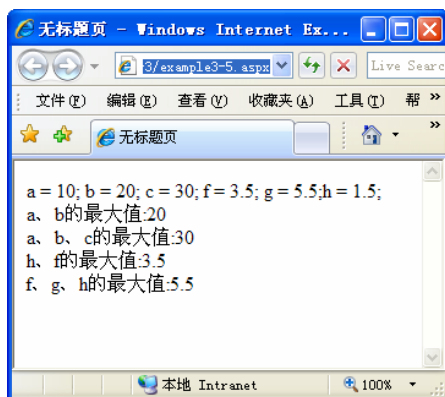


图 3.6 example3-5 网页运行的结果

从【例 3-5】可以看出，max 方法是求若干参数中的最大值，类 Myclass 中有 4 个同名的 max 方法，它们或参数个数不一样，或参数类型不一样。在调用 max 方法时，编译器会根据调用时给出的实参个数及类型调用相应的方法。

### 3.3 属性

为了实现良好的数据封装和数据隐藏，类的字段成员的访问属性一般设置成 private 或 protected，这样在类的外部就不能直接读/写这些字段成员了，通常的办法是提供 public 级的方法来访问私有的或受保护的字段。

但 C# 提供了属性（property）这个更好的方法，把字段域和访问它们的方法相结合。对类的用户而言，属性值的读/写与字段域语法相同；对编译器来说，属性值的读/写是通过类中封装的特别方法 get 访问器和 set 访问器实现的。属性的声明方法如下：

语法形式：

[属性集信息] [属性修饰符] 类型 成员名

```
{
    访问器声明
}
```

其中：

- 属性修饰符：与方法修饰符相同，包括 `new`、`static`、`virtual`、`abstract`、`override` 和 4 种访问修饰符的合法组合，它们遵循相同的规则。
- 类型：指定该声明所引入的属性的类型。
- 成员名：指定该属性的名称。
- 访问器声明：声明属性的访问器，可以是一个 `get` 访问器或一个 `set` 访问器，或者两个都有。

语法形式：

```
get                // 读访问器
{    ...    }      // 访问器语句块
set                // 写访问器
{    ...    }      // 访问器语句块
```

`get` 访问器的返回值类型与属性的类型相同，所以语句块中的 `return` 语句必须有一个可隐式转换为属性类型的表达式。`set` 访问器没有返回值，但它有一个隐式的值参数，其名称为 `value`，它的类型与属性的类型相同。

同时包含 `get` 和 `set` 访问器的属性是读/写属性，只包含 `get` 访问器的属性是只读属性，只包含 `set` 访问器的属性是只写属性。

**【例 3-6】** 前面学习了 `TextBox` 控件的使用，包括属性的使用。下面声明 `TextBox` 类，用于体现属性的使用。

设计步骤如下：

- ① 在 `Chapter3` 网站上添加一个命名为“`example3-6`”的网页。
- ② 在“`example3-6.aspx.cs`”代码页中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    TextBox textbox1 = new TextBox();
    textbox1.Text = "这是 textbox1";
    Response.Write("textbox1 的 text 属性值: "+textbox1.Text);
    Response.Write("<br>textbox1 的 fontsize 属性值: " + textbox1.FontSize);
    textbox1.Multiline = true;
}
class TextBox
{
    private string text;
    private int fontsize;
    private bool multiline;
    public TextBox()
    {
        text = "text1";
```

```

        fontsize = 12;
        multiline = false;
    }
    public string Text
    {
        //Text 属性，可读可写
        get
        { return text; }
        set
        { text = value; }
    }
    public int FontSize
    {
        //FontName 属性，只读属性
        get
        { return fontsize; }
    }
    public bool MultiLine
    {
        //MultiLine 属性，只写
        set
        { multiline = value; }
    }
}

```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.7 所示。

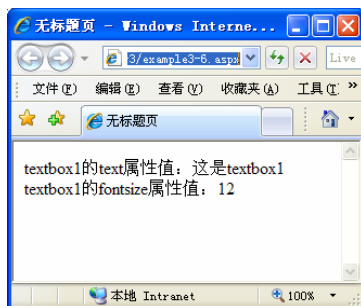


图 3.7 example3-6 网页运行的结果

在这个示例中，编译器根据属性出现的位置调用不同的访问器。如果在表达式中引用该属性则调用 `get` 访问器，如果给属性赋值则调用 `set` 访问器，赋值号右边的表达式值传给 `value`。

**注意：**尽管属性与字段域有相同的使用语法，但属性本身并不代表字段域。属性不直接对应存储位置，所以不能把它当变量使用，不能把属性作为 `ref` 或者 `out` 参数传递。属性和方法一样，也有静态修饰；在静态属性的访问器中不能访问静态数据也不能引用 `this`。

## 3.4 继承和多态

继承是面向对象语言的基本特征，是实现代码复用的手段。继承使得在原有的类基础之上对原有程序进行扩展，从而提高程序开发的速度，实现代码的复用。同一种方法作用于不



同对象可以产生不同的结果，这就是多态性。它是在基类中使用虚方法、在其派生类中使用重载实现的。

### 3.4.1 继承

继承不但可以使用现有类的所有功能，而且还可以对这些功能进行扩展。继承产生的类称为派生类或子类，而被继承的类则称为基类、超类或父类。客观世界中的许多事物之间往往都具有相同的特征，具有继承的特点。图 3.8 所示是采用类的层次图表示继承的例子。

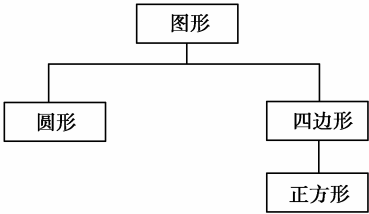


图 3.8 类的层次结构示例

#### 1. 继承的实现

语法形式：

```
[属性集信息] [类修饰符] class 派生类: 基类
{
    [派生类成员]
}
```

其中，属性集信息、类修饰符的使用参见类的声明。  
例如：

```
public class Shape //定义图形类 Shape
{
    protected string Color;
    public Shape() //无参构造函数
    { }
    public Shape(string Color) //构造函数
    { this.Color = Color; }
}
public class Circle : Shape //定义圆类 Circle，从 Shape 类中派生
{
    private double Radius;
    public Circle(string Color, double Radius)
    {
        this.Color = Color;
        this.Radius = Radius;
    }
}
```

## 2. base关键字

实际上，上述程序中 `Circle` 调用的是父类（`Shape`）的无参构造函数，而不是有参构造函数，它等同于：

```
public class Circle : Shape //定义圆类 Circle，从 Shape 类中派生
{
    private double Radius;
    public Circle(string Color, double Radius):base()
    {
        this.Color = Color;
        this.Radius = Radius;
    }
}
```

上述程序中的 `Circle` 类也可以改写成：

```
public class Circle: Shape
{
    private double Radius;
    public Circle(string Color,double Radius):base(Color)
    {
        this.Color = Color;
        this.Radius = Radius;
    }
}
```

关键字 `base` 的作用是调用 `Shape` 类的构造函数，并将 `Circle` 类的变量初始化。`base` 关键字除了能调用基类对象的构造函数，还可以调用基类的方法。

## 3. System.Object类

C#的所有类都派生于 `System.Object` 类。在定义类时如果没有指定派生于哪一个类，则系统默认其派生于 `Object` 类。上述程序中 `Shape` 的定义等同于：

```
public class Shape :System.Object
{
    //TODO...
}
```

`System.Object` 类常见的公有方法如下。

- (1) `Equals`：如果两个对象具有相同值，则方法将返回 `true`。
- (2) `GetHashCode`：方法返回对象值的散列码。
- (3) `ToString`：通过在派生类中重写该方法，返回一个表示对象状态的字符串。
- (4) `GetType`：根据内置对象类型指针获得当前对象的实际类型。

**【例 3-7】** 在网站 `Chapter3` 上添加一个网页，命名为 `example3-7`，用于体现类的继承的使用。

设计步骤如下：

- ① 在 Chapter3 网站上添加一个命名为 “example3-7” 的网页。
- ② 在 “example3-7.aspx.cs” 代码页中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    Circle C = new Circle("Red",32);           //实例化一个 Circle 类 C
    Response.Write("圆的颜色: "+C.GetColor()+" 圆的面积: "+C.GetArea ());
    Rectangular R = new Rectangular("Bule",23,43);//实例化一个 Rectangular 类 R
    Response.Write("<br>矩形的颜色: "+R.GetColor()+" 矩形的面积: "+R.RectangularAre());
}
public class Shape : System.Object             //定义 Shape 类，从 Object 类中派生
{
    protected string Color;
    public Shape()
    { }
    public Shape(string Color)
    { this.Color = Color;    }
    public string GetColor()
    { return Color;    }
}
public class Circle : Shape                    //定义 Circle 类，从 Shape 类中派生
{
    private double Radius;
    public Circle(string Color, double Radius):base(Color)
    {
        this.Color = Color;
        this.Radius = Radius;
    }
    public double GetArea ()
    { return System.Math.PI * Radius * Radius;    }
}
public class Rectangular : Shape              // 派生类 Rectangular，从 Shape 类中派生
{
    protected double Length, Width;
    public Rectangular()
    { Length = Width = 0;    }
    //未使用 base 关键字直接调用 Shape 类的无参构造函数
    public Rectangular(string Color, double Length, double Width)
    {
        this.Color = Color;
        this.Length = Length;
        this.Width = Width;
    }
    public double RectangularAre()
    { return Length * Width;    }
```

```

public double PerimeterIs()//周长
{
    return (2 * (Length + Width));
}

```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.9 所示。

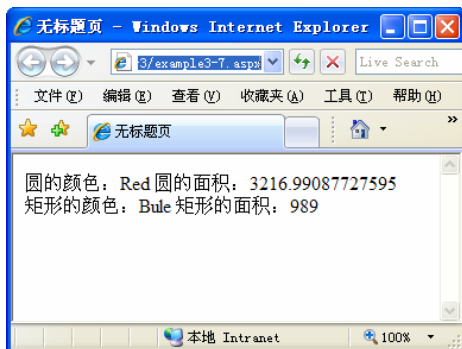


图 3.9 example3-7 网页运行的结果

因为基类的字段 Color 的修饰符为 protected，方法 GetColor 的修饰符为 public，所以它们在派生类中都可使用。派生类在继承基类成员的基础上，也可以各自增加功能。

### 3.4.2 多态

多态也是面向对象语言的基本特征之一，指在程序执行之前无法根据函数名和参数确定调用哪一个操作，而在程序执行过程中，根据实际运行情况动态确定，从而带来编程的高度灵活性。使用虚方法可以实现多态。

#### 1. 虚方法的重载

在类的方法前加上关键字 virtual，则该方法被称为虚方法。通过对虚方法的重载，实现在程序运行过程中确定调用的方法。需要注意的是，这里所讲的重载与前面所讲的通过参数类型与参数个数的不同实现的重载的含义是不同的。

例如，在【例 3-7】中计算圆形的面积时用了方法 GetArea，也可以通过虚方法实现。在 Shape 中增加虚方法：

```

public virtual double GetArea ()
{
    return 0.0;
}

```

在 Circle 类中对虚方法重载：

```

public override double GetArea ()
{
    return System.Math.PI * radius * radius;
}

```

#### 2. 抽象类和抽象方法

抽象类是一种特殊的基类，并不与具体的事物相联系。抽象类的定义使用关键字 abstract。在图 3.8 所示类的层次结构中，并没有“图形”这种具体的事物，所以将“图形”定义为抽象类，派生了“圆形”和“四边形”这样一些可以产生具体实例化的普通类。需要注意的是，抽象类不能被实例化，它只能作为其他类的基类。将 Shape 类定义为抽象类：

```
public abstract class Shape //定义抽象类 Shape
{ //TODO... }
```

在抽象类中也可以使用关键字 **abstract** 定义抽象方法，要求所有的派生非抽象类都要重载实现抽象方法。引入抽象方法的原因在于，抽象类本身是一种抽象的概念，有的方法并不要求具体的实现，而是留下来让派生类来重载实现的。**Shape** 类中计算具体面积的方法本身没有什么具体的意义，而只有到了派生类 **Circle** 类和 **Rectangular** 类才可以计算具体的面积。抽象方法的写法为

```
public abstract double GetArea(); //抽象类中定义抽象方法 GetArea，计算面积
```

则派生类重载实现为

```
public override double GetArea() //在派生类中重载抽象方法 GetArea，计算面积
{ //TODO... }
```

### 3. 密封类和密封方法

抽象类只能作为基类，由其他类继承，不能被实例化。相对应的有一种不能被其他类继承的类，叫密封类，使用 **sealed** 关键字定义。如将 **Rectangular** 类定义为密封类：

```
public sealed class Rectangular:Shape //定义密封类 Rectangular
{ //TODO... }
```

这样 **Rectangular** 类的派生类 **Square** 就不再保留，否则就会出错。如果类的方法声明包含 **sealed** 修饰符，则称该方法为密封方法。若类的实例方法声明包含 **sealed** 修饰符，则必须同时使用 **override** 修饰符。使用密封方法可以防止派生类进一步重写该方法。如果将圆形 **Circle** 类的 **GetArea** 方法定义为密封类，则必须先将 **Shape** 类的 **GetArea** 方法定义为：

```
public virtual double GetArea()
{ //TODO... }
```

然后在 **Circle** 类中实现密封方法：

```
public sealed override double GetArea()
{ //TODO... }
```

**【例 3-8】** 在网站 Chapter3 上添加一个网页，命名为 example3-8，用于体现类的多态。设计步骤如下：

- ① 在 Chapter3 网站上添加一个命名为“example3-8”的网页。
- ② 在“example3-8.aspx.cs”代码页中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    Circle C = new Circle("Red",54);
    Response.Write("圆的面积: " + C.GetArea() + " 圆的颜色: "+C.GetColor());
    Rectangular R = new Rectangular("Blue",43,65);
    Response.Write("<br>矩形的面积: "+R.GetArea()+ " 矩形的颜色: "+R.GetColor()+ "矩形的
```

```

        周长"+R.PerimeterIs());
    }
public abstract class Shape : System.Object           //定义抽象类 Shape，从 Object 类中派生
{
    protected string Color;
    public Shape()
    {    }
    public Shape(string Color)
    {    this.Color = Color;    }
    public abstract string GetColor();                //定义抽象方法，没有主体 "{}"
    public virtual double GetArea()                   //定义虚方法
    {    return 0.0;    }
}
public class Circle : Shape                           //定义 Circle 类，从 Shape 类中派生
{
    private double Radius;
    public Circle(string Color, double Radius): base(Color)
    {
        this.Color = Color;
        this.Radius = Radius;
    }
    public override double GetArea()                  //对虚方法进行重载
    {    return System.Math.PI * Radius * Radius;    }
    public override string GetColor()                 //对抽象方法进行重载
    {    return this.Color;    }
}
public sealed class Rectangular : Shape               //密封类 Rectangular，从 Shape 类中派生，
                                                        但不能被继承
{
    protected double Length, Width;
    public Rectangular()
    { Length = Width = 0; }
    //未使用 base 关键字直接调用 Shape 类的无参构造函数
    public Rectangular(string Color, double Length, double Width)
    {
        this.Color = Color;
        this.Length = Length;
        this.Width = Width;
    }
    public sealed override double GetArea()            //定义密封方法，对虚方法进行重载
    {    return Length * Width;    }
    public override string GetColor()                 //必须对抽象方法 GetColor 进行重载
    {    return this.Color;    }
    public double PerimeterIs()                       //计算周长
    {    return (2 * (Length + Width));    }
}

```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.10 所示。

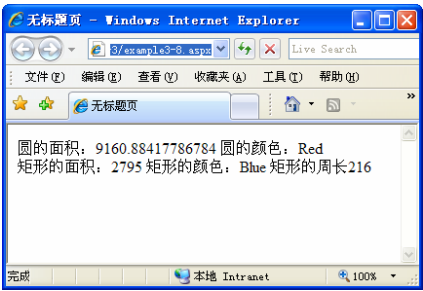


图 3.10 example3-8 网页运行的结果

## 3.5 委托和事件

### 3.5.1 委托

C#的委托相当于 C/C++中的函数指针。函数指针用指针获取一个函数的入口地址，实现对函数的操作。委托与 C/C++中的函数指针的不同之处在于，委托是面向对象的，是引用类型，因此对委托的使用要“先定义，后实例化，最后才调用”。

定义委托使用关键字 `delegate`：

```
delegate int SomeDelegate(int nID, string sName);
```

再实例化：

```
SomeDelegate d1 = new SomeDelegate(wr.InstanceMethod);
```

最后调用：

```
d1(5, "aaa");
```

通过委托 `SomeDelegate` 实现对方法 `InstanceMethod` 的调用，调用还必须有一个前提条件：方法 `InstanceMethod` 有参数且和定义 `SomeDelegate` 的参数一致，并且返回值为 `int`。方法 `InstanceMethod` 的定义：

```
public int InstanceMethod(int nID, string sName)
```

委托的实例化中的参数既可以是实例方法，也可以是静态方法。

**【例 3-9】** 在网站 Chapter3 上添加一个网页，命名为 `example3-9`，用于体现委托的使用。设计步骤如下：

- ① 在 Chapter3 网站上添加一个命名为“`example3-9`”的网页。
- ② 在“`example3-9.aspx.cs`”代码页中添加如下代码：

```
delegate string SomeDelegate(); //定义委托 SomeDelegate
protected void Page_Load(object sender, EventArgs e)
{
```

```

A a=new A();                                //实例化类 A
//实例化委托 SomeDelegate, 也可以是: SomeDelegate S1 = a.ExampleMethod;
SomeDelegate S1 = new SomeDelegate(a.ExampleMethod);
Response.Write(S1());                       //调用委托, 返回的结果显示在页面上
SomeDelegate S2 = A.StaticMethod();         //关联到静态方法
Response.Write("<br>" + S2());

}
class A
{
    public string ExampleMethod ()           //实例方法
    { return "调用实例方法 ExampleMethod"; }
    public static string StaticMethod()      //静态方法
    { return "调用静态方法 StaticMethod"; }
}

```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.11 所示。

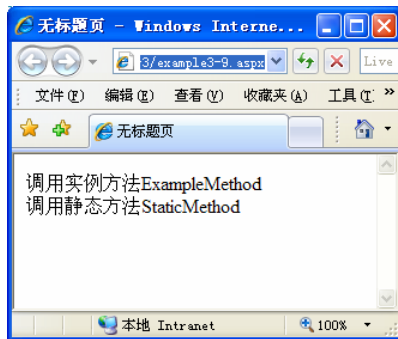


图 3.11 example3-9 网页运行的结果

### 3.5.2 事件

事件为类和类的实例定义发出通知，从而将事件和可执行代码捆绑在一起。事件最常见的用途是窗体编程，当发生单击按钮、移动鼠标等事件时，相应的程序将收到通知，再执行代码。

C#事件是按“发布-预订”方式工作的。先在一个类中发布事件，然后在任意数量的类中对事件进行预订。事件的工作过程如图 3.12 所示。

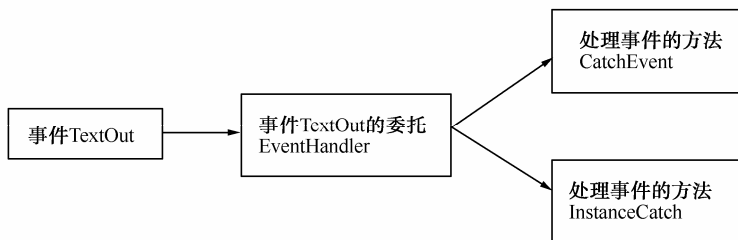


图 3.12 事件的工作过程

C#事件机制是基于委托实现的，因此首先要定义一个委托 EventHandler:



```
public delegate void EventHandler(object from, myEventArgs e)
```

其中：

■ **myEventArgs** 类：派生于 **EventArgs** 类（包含事件数据的类的基类），实现自定义事件数据的功能。

■ **from**：表示发生事件的对象。

定义事件格式为：

```
event 事件的委托名 事件名
```

例如，事件 **TextOut** 定义为

```
public event EventHandler TextOut;
```

事件的激活一般写成：

```
if (TextOut != null)
{
    TextOut(this,new EventArgs())
};
```

检查 **TextOut** 事件有没有被订阅，若不为 **null**，则表示有用户订阅。订阅事件的是 **TestApp** 类，首先实例化 **EventSource**，然后订阅事件：

```
evsrc.TextOut += new EventHandler(CatchEvent);
```

也可以取消订阅：

```
evsrc.TextOut -= new EventHandler(CatchEvent);
```

方法 **evsrc.TriggerEvent()** 激活事件，如果已经订阅了事件，则调用处理代码，否则什么也不执行。这里要注意，**CatchEvent** 和 **InstanceCatch** 方法的签名与定义的委托 **EventHandler** 的签名要相同，这与委托的工作机制类似。

**【例 3-10】** 在网站 **Chapter3** 上添加一个网页，命名为 **example3-10**，用于体现委托的使用。

设计步骤如下：

- ① 在 **Chapter3** 网站上添加一个命名为“**example3-10**”的网页。
- ② 在“**example3-10.aspx.cs**”代码页中添加如下代码：

```
public delegate void CalculateEventHandler(object sender, EventArgs e); //定义委托 CalculateEvent
                                                                    Handler
protected void Page_Load(object sender, EventArgs e) //加载页面时所要执行的方法
{
    CalculateWithEvent CalculateCase=new CalculateWithEvent(); //实例化 CalculateWith
                                                                    Event 类
    CalculateCase.Calculate += new CalculateEventHandler(CalculateCase.Calculate);
                                                                    //订阅事件
    Response.Write(CalculateCase.Add(5,6)); //调用方法并在页面上显示结果
}
protected void CalculateCase_Calculate(object sender, EventArgs e) //委托所要调用的方法
```

```

{    Response.Write("运算事件被调用<br>");    }
public class CalculateWithEvent
{
    public event CalculateEventHandler Calculate;        //定义事件 Calculate
    protected void OnCalculate(EventArgs e)
    {
        if (Calculate != null)
        {    Calculate(this, e);    }                    //引发事件
    }
    public string Add(int a, int b)
    {
        OnCalculate(EventArgs.Empty);
        return "a+b="+(a+b).ToString();                //把 a+b 的结果转换为 string 类型
                                                         并返回
    }
}

```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.13 所示。

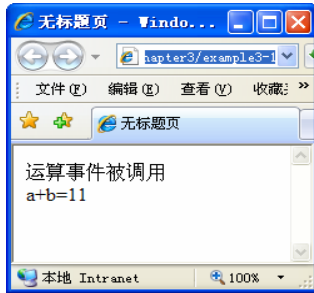


图 3.13 example3-10 网页运行的结果

## 3.6 接口

接口是用来定义一种程序的协定。接口好比一种模板，这种模板定义了实现接口的对象必须实现的方法，其目的就是让这些方法可以作为接口实例被引用。接口的定义如下：

```

public interface IPartA
{    void SetDataA(string dataA);    }

```

接口使用关键字 `interface` 定义，接口可以使用的修饰符包括 `new`、`public`、`protected`、`internal`、`private` 等。接口的命名通常以 `I` 开头，如 `IPartA`、`IPartB`。接口的成员可以是方法、属性、索引器和事件，但不可以有任意的成员变量，也不能在接口中实现接口成员。接口不能被实例化。接口的成员默认是公共的，因此不允许成员加上修饰符。

**【例 3-11】** 在网站 Chapter3 上添加一个网页，命名为 `example3-11`，用于体现接口的使用。

设计步骤如下：

① 在 Chapter3 网站上添加一个命名为“`example3-11`”的网页。

② 在 “example3-11.aspx.cs” 代码页中添加如下代码：

```
public partial class example3_11 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        SharedClass a = new SharedClass();
        Response.Write(a.SetDataA("接口 IPartA")+"<br>");
        Response.Write(a.SetDataB("接口 IPartB"));
    }
}

public interface IPartA                                // 定义接口 IPartA
{
    string SetDataA(string dataA);    }

public interface IPartB : IPartA                // 定义接口 IPartB，继承 IPartA
{
    string SetDataB(string dataB);    }

public class SharedClass : IPartB                // 定义类 SharedClass，继承接口 IPartB
{
    public string SetDataA(string dataA)          // 实现接口 IPartA 的方法 SetDataA
    {
        return dataA;    }

    public string SetDataB(string dataB)          // 实现接口 IPartB 的方法 SetDataB
    {
        return dataB;    }
}
```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.14 所示。



图 3.14 example3-11 网页运行的结果

### 3.7 集合

集合是一组组合在一起的类似的类型化对象。集合基于 `ICollection` 接口、 `IList` 接口和 `IDictionary` 接口，其中  `IList` 接口和 `IDictionary` 接口都是从 `ICollection` 接口派生的。因此，所有集合都直接或间接地基于 `ICollection` 接口。常用的基于  `IList` 接口的集合有 `Array`、`ArrayList` 和 `List`。基于 `ICollection` 接口的集合有 `Queue`、`Stack` 和 `LinkedList`。在这些集合中，每个元素都只包含一个值。常用的基于 `IDictionary` 接口的集合有 `Hashtable` 和 `SortedList`，这些集合的每个元素都包含一个键和一个值。

3.7.1 使用Array类进行排序与查找

Array 类是用于对数组进行排序和搜索的类。Array 类提供 Sort()和 BinarySearch()方法，用于排序与查找。另外，它还提供 Reverse()方法进行反排序。

1. Array.Sort()与Reverse()方法

Array.Sort()方法可以实现对一维数组的排序，其常用的几种形式如表 3.4 所示。

表 3.4 Array.Sort()方法常用的几种形式

形 式	说 明
Array.Sort (Array)	使用 Array 数组中每个元素的 Icomparable 接口实现，对整个一维 Array 数组元素排序
Array.Sort(Array,Array)	基于第一个 Array 数组中的关键字，使用每个关键字的 Icomparable 接口实现，对两个一维 Array 数组对象排序
Array.Sort(Array, IComparer)	使用指定的 Icomparer 接口，对一维 Array 数组元素排序
Array.Sort(Array,Array,IComparer)	基于第一个 Array 数组中的关键字，使用指定的 Icomparer 接口，对两个一维 Array 数组对象排序

Array.Reverse()方法可以用来对整个数组的顺序进行反转，其形式如下：

```
public static void Reverse(Array);
```

2. Array.BinarySearch()方法

Array.BinarySearch()方法实现在已经排序的一维数组中查找元素，其常用的几种形式如表 3.5 所示。

表 3.5 Array. BinarySearch()方法常用的几种形式

形 式	说 明
Array.BinarySearch(Array, Object)	使用由 Array 数组中每个元素和指定的对象实现的 IComparable 接口，在整个一维排序 Array 数组中搜索特定元素
Array.BinarySearch(Array,Object, IComparer)	使用指定的 IComparer 接口，在整个一维排序 Array 数组中搜索值

**注意：**在执行 BinarySearch()之前必须先对数组进行排序。

**【例 3-12】** 在网站 Chapter3 上添加一个网页，命名为 example3-12，用于体现 Array 类的使用。

设计步骤如下：

- ① 在 Chapter3 网站上添加一个命名为“example3-12”的网页。
- ② 在“example3-12.aspx.cs”代码页中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
```

```
int[] NumArray = { 34,65,68,12,45,23,54,33,98}; //定义一个数组
Show(NumArray);                                //未排序时数组顺序
Array.Sort(NumArray);                          //调用 Sort 方法排序
Show(NumArray);                                //排序后的数组顺序
int i = Array.BinarySearch(NumArray, 33)+1;     //查找 33 所在的位置
Response.Write("33 在第"+i+"位<br>");
Array.Reverse(NumArray);                       //调用 Reverse 方法反转数组
Show(NumArray);                                //数组反转后的数组顺序
}
protected void Show(int[] arr)
{
    foreach (int intobj in arr)                //遍历数组
    {      Response.Write(intobj + " ");      }
    Response.Write("<br>");                    //换行
}
```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.15 所示。



图 3.15 example3-12 网页运行的结果

### 3.7.2 使用Stack类

Stack 类表示对象的“后进先出”集合。Stack 类的常用方法如表 3.6 所示。

表 3.6 Stack 类的常用方法

方 法	说 明
Clear	从 Stack 中移除所有对象
Pop	移除并返回位于 Stack 顶部的对象
Push	将对象插入 Stack 的顶部
Peek	返回位于 Stack 顶部的对象但不将其移除

Stack 类将其对象存储在数组中。只要数组足够大到可以存储新的对象，调用 Push 方法就是非常有效的。但是如果内部数组必须调整大小，就必须分配新数组并把现有的对象复制到新数组中。为了避免这一昂贵的操作成本，可以预选分配一个大的内部数组，或定义满足

执行需要的合适的增长系数。

**【例 3-13】** 在网站 Chapter3 上添加一个网页，命名为 example3-13，用于体现 Stack 类的使用。

设计步骤如下：

- ① 在 Chapter3 网站上添加一个命名为 “example3-13” 的网页。
- ② 在 “example3-13.aspx.cs” 代码页中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    //创建并初始化栈
    Stack myStack = new Stack();
    myStack.Push("Hello");
    myStack.Push("World");
    myStack.Push("!");
    // 显示栈的元素数目和其中的元素
    Response.Write("栈 myStack 中的元素数目: "+myStack.Count);
    Response.Write(", 元素有: ");
    PrintValues(myStack);
    //调用一次 Pop()方法后显示栈的元素数目和其中的元素
    myStack.Pop();
    Response.Write("<br>调用一次 Pop()方法后栈中的元素数目: " + myStack.Count);
    Response.Write(", 元素有: ");
    PrintValues(myStack);
}
public void PrintValues(ICollection myCollection)
{
    foreach (Object obj in myCollection)           //遍历元素并显示在页面上
        Response.Write(" "+obj);
    Response.Write("<br>");
}
```

- ③ 按 **【Ctrl+F5】** 组合键运行网页，结果如图 3.16 所示。

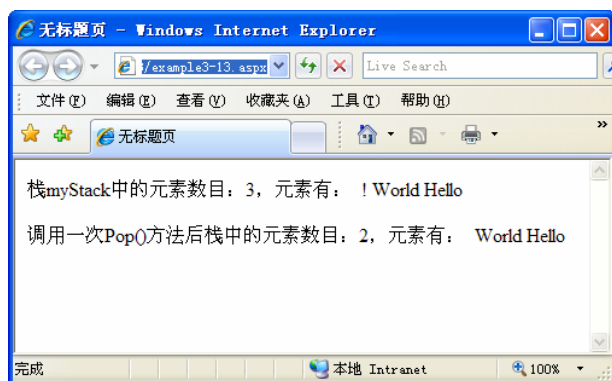


图 3.16 example3-13 网页运行的结果

## 3.8 命名空间和局部类

### 3.8.1 命名空间

.NET Framework 使用命名空间来组织众多的类，它是类的逻辑分类。当需要使用某个类时，可以使用带有命名空间的完全限定名。例如：

```
System.Console.WriteLine("中国");
```

其中，**System** 是一个命名空间，**Console** 是该命名空间中包含的类，**WriteLine** 是该类的一个方法。如果使用 **using** 关键字引入了命名空间，则不必使用完全限定名。例如：

```
using System
Console.WriteLine("中国");
```

上面引用了系统的命名空间，当然，也可以使用 **namespace** 关键字来定义自己的命名空间。在编写大规模系统时，命名空间可用于帮助控制范围、组织代码。

**【例 3-14】** 在网站 Chapter3 上添加一个网页，命名为 **example3-14**，用于体现命名空间的使用。

设计步骤如下：

- ① 在 Chapter3 网站上添加一个命名为“**example3-14**”的网页。
- ② 在“**example3-14.aspx.cs**”代码页中添加如下代码：

```
namespace myNamespace                                //自定义命名空间 myNamespace
{
    class myClass
    {
        public string sampleMethod()
        { return "爱我中华"; }
    }
}
public partial class example3_14 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        myNamespace.myClass c = new myNamespace.myClass();//带有命名空间的完全限定名
        Response.Write(c.sampleMethod());
    }
}
```

- ③ 按 **【Ctrl+F5】** 组合键运行网页，结果如图 3.17 所示。



图 3.17 example3-14 网页运行的结果

上述代码定义了一个自定义的命名空间 `myNamespace`，然后声明了另一个类 `myProgram`，类 `myProgram` 中引用了 `myNamespace` 命名空间。也可以使用 `using` 来引用命名空间，如下所示：

```
using myNamespace;                                //引用命名空间 myNamespace
namespace myNamespace                             //自定义命名空间 myNamespace
{
    class myClass
    {
        public string sampleMethod()
        { return "爱我中华"; }
    }
}
public partial class example3_14 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        myClass c = new myClass();                //直接实例化
        Response.Write(c.sampleMethod());
    }
}
```

3.8.2 局部类

从 C#2.0 开始，C# 提供了一种新的技术特性——局部类。局部类允许将类、结构、接口的定义和实现代码分成小块，并且分开放置在多个源文件中。很显然，这种方案能够使得代码容易开发和维护。另外，局部类可以分离代码生成器产生的代码和开发人员书写的代码，这使得利用工具来加强产生的代码变得容易。实现局部类，即在多个部分中定义一个类时，必须使用新的 `partial` 关键字来标注分割；这样可以将局部类的一部分放在一个源文件中，而将局部类的另一部分放在不同的源文件中。例如，前面所介绍的例子中的代码几乎都放在下面的局部类中。

```
public partial class exampleX_X : System.Web.UI.Page    //X_X 表示例号
{
    .....
}
```

在下面的示例中，一个类的实现代码被分成两个部分，放置在两个不同的源文件中。另外，代码的第一部分是由代码产生器生成的，而第二部分是开发人员手工编写的。第一部分



代码如下：

```
public partial class MyClass
{
    private string name;
    public MyClass(string n)
    {
        .....
    }
}
```

第二部分代码如下：

```
public partial class MyClass
{
    public void SubmitOrder()
    {
        .....
    }
}
```

第一部分代码主要包括程序代码的结构；第二部分代码是手工编写，主要包括一些业务逻辑的实现。由于二者都使用 `partial` 关键字和相同的类名 `MyClass`，因此，两个部分都称为局部类 `MyClass`。这种实现方法在保证代码结构的同时，使代码更加清晰和易于维护。

**【例 3-15】** 在网站 Chapter3 上添加一个网页，命名为 `example3-15`，用于体现局部类的使用。

设计步骤如下：

- ① 在 Chapter3 网站上添加一个命名为“`example3-15`”的网页。
- ② 在“`example3-15.aspx.cs`”代码页中添加如下代码：

```
public partial class MyClass //定义局部类 MyClass，存放构造函数
{
    private string name;
    private int age;
    public MyClass(string n,int a) //MyClass 的构造函数
    {
        name = n;
        age = a;
    }
}

public partial class MyClass //定义局部类 MyClass，存放获得姓名函数 GetName()
{
    public string GetName()
    {
        return name;
    }
}

public partial class MyClass //定义局部类 MyClass，存放获得姓名函数 GetAge()
{
    public int GetAge()
    {
        return age;
    }
}

public partial class example3_15 : System.Web.UI.Page
```

```

{
    protected void Page_Load(object sender, EventArgs e) //加载页面时所执行的方法
    {
        MyClass M = new MyClass("王明",27);           //实例化一个 MyClass 对象 M
        Response.Write("姓名: "+M.GetName());         //在页面上输出姓名
        Response.Write("<br>年龄: "+M.GetAge());        //在页面上输出年龄
    }
}

```

③ 按【Ctrl+F5】组合键运行网页，结果如图 3.18 所示。



图 3.18 example3-15 网页运行的结果

## 习 题

1. C#语言是一种面向对象的程序设计语言，拥有面向对象语言的三大特点，即封装性、继承性和\_\_\_\_\_。
2. 接口成员是否一定是公共的？
3. 类的成员默认的访问修饰符是（ ）。  
A. public      B. private      C. protected      D. internal
4. 以下修饰符中，（ ）必须由派生类实现。  
A. private      B. sealed      C. abstrect      D. final
5. 创建一个基类，然后进行继承练习。例如，学校包括本科生和研究生，分别创建一个学生类和一个研究生类，研究生类从学生类派生。

## 第 4 章 ASP.NET 应用程序基础与内置对象

第 2 章和第 3 章介绍了 C# 的语法知识，但是，仅仅熟悉了 C# 语法知识还无法进行 ASP.NET 应用程序的开发，还需进一步了解 ASP.NET 应用程序结构和页面等相关知识，对此有一定了解后才能真正进入网站开发阶段。本章将介绍 ASP.NET 应用程序基础及 ASP.NET 内置对象等相关知识。

### 4.1 ASP.NET 应用程序基础

ASP.NET 提供了更多增强的应用程序类型，其中包括 Web 应用程序、移动 Web 应用程序和 Web 服务。与普通的 Windows 应用程序不同，ASP.NET 将所有文件、页面、处理程序、模块和可执行代码的总和定义为 ASP.NET 应用程序。

#### 4.1.1 aspx 代码模式和页面元素

在 ASP.NET 中，代码可以用两种模式存储：一种是单文件页模型，另一种是代码隐藏页模型。在代码隐藏页模型中，显示信息的代码与逻辑处理的代码分别放在不同的文件中；在单文件页模型中，两种代码放置在同一个文件中。第 2 章、第 3 章中所有的例子都是代码隐藏页模型。

推荐大家使用代码隐藏页模型，因为这种模型代码结构清晰且便于调试和维护，符合大型项目的代码规范要求。

**【例 4-1】** 新建一个网站，在网站中添加一个单文件页模型网页，命名为 example4-1。设计步骤如下：

① 在 D 盘新建一个命名为“Chapter4”的文件夹，打开 VS 2008，依次展开工具栏中“新建”→“网站”选项，系统弹出“新建网站”对话框，选择“ASP.NET 网站”模板，语言选择“Visual C#”，框架选择“.NET Framework 3.5”，位置选择“D:\Chapter4”，单击**【确定】**按钮。

② 打开“资源管理器”→右击“D:\Chapter4\”→单击“添加新项”选项，系统弹出“添加新项”对话框，选择“Web 窗体”并且重新命名为“example4-1”。取消对话框右下角的“将代码放在单独的文件中”复选框选中状态，如图 4.1 所示。单击**【添加】**按钮，系统添加了一个单文件页并切换到源视图。

源视图中代码如下所示：

```
<% @ Page Language="C#" %> //页面指令
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server"></script> //代码脚本块
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

```
<title>无标题页</title>
</head>
<body>
  <form id="form1" runat="server">
    <div></div>
  </form>
</body>
</html>
```



图 4.1 添加一个单文件页模型网页

其中，页面指令是以<%@ ..... %>括起来的代码，页面指令用于指定当前页编译处理时所使用的设置，一个页面可包含多条页面指令。指令的大小写并不重要，也不要求在属性值的两侧加上引号。

代码脚本块是由<script runat="server"></script>标签对括起来的程序代码。在代码脚本块中可以定义页面的全局变量及程序处理过程等。

页面内容的形式与标准 HTML 页面基本一致。除了标准页面所具有的元素外，还包含一个 form 元素。

需要注意的是，不是所有的 aspx 网页文件都包含代码脚本块部分。在单文件页模型中（上例即为单文件页模型），页的标记及其编程代码位于同一个.aspx 文件中，编程代码位于 script 块中，该块包含 runat="server"属性，此属性将其标记为 ASP.NET 应执行的代码。而代码隐藏页模型，可以在一个文件（.aspx 文件）中保留标记，并在另一个文件中保留编程代码。代码文件的名称会根据所使用的编程语言而有所变化。在代码隐藏页模型中，.aspx 文件中不存在具有 runat="server"属性的 script 块。

4.1.2 页面指令

页面指令指定一些设置，页和用户控件编译器在处理 ASP.NET Web 窗体页（.aspx 文件）和用户控件（.ascx）文件时使用这些设置。使用指令时，标准的做法是将指令放置于文件的顶端。每个指令都包含一个或多个属性与值，形式如下：

```
<%@ 指令 属性 1="值" ..... %>
```

表 4.1 列出了 ASP.NET 提供的页面指令。

表 4.1 ASP.NET 页面指令

指 令	说 明
@ Assembly	以声明方式将程序集链接到当前页或用户控件
@ Control	定义 ASP.NET 页分析器和编译器使用的控件特定的属性；只能包含在.ascx 文件（用户控件）中
@ Implements	以声明方式指示页或用户控件实现指定的.NET Framework 接口
@ Import	将命名空间显式导入页或用户控件中
@ Master	将页标识为母版页，并定义 ASP.NET 页分析器和编译器使用的属性；只能包含在.master 文件中
@ MasterType	定义用于确定页的 Master 属性类型的类或虚拟目录
@ OutputCache	以声明方式控制页或用户控件的输出缓存策略
@ Page	定义 ASP.NET 页分析器和编译器使用的页特定的属性；只能包含在.aspx 文件中
@ PreviousPageType	创建一个强类型的引用，该引用指向来自跨页发送的目标的源页
@ Reference	以声明方式将页、用户控件或 COM 控件链接到当前的页或用户控件
@ Register	将别名与命名空间和类相关联，以便在用户控件和自定义服务器控件被纳入到请求页或用户控件中时得以呈现

1. Page指令

Page 指令用于定义特定于页面的属性，ASP.NET 页分析器和编译器根据此属性来编译页面。Page 指令只能置于.aspx 文件中，并且一个页面只允许出现一条 Page 指令。例如：

```
<% @ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default"%>
```

其中，Language 指定页面的内联代码的编程语言为 C#；AutoEventWireup 指示页的事件是否自动绑定；CodeFile 指定与页关联的代码隐藏文件的路径；Inherits 与 CodeFile 属性一起使用，后者包含指向代码隐藏类的源文件的路径。

2. Import指令

Import 指令的功能是将命名空间显式导入到 ASP.NET 应用程序文件（如网页、用户控件、母版页或 Global.asax 文件）中，同时使导入的命名空间的所有类和接口可用于文件。导入的命名空间可以是 .NET Framework 类库或用户定义的命名空间的一部分。例如：

```
<% @ Import Namespace="value" %>
```

其中，value 表示命名空间名称。导入命名空间后，用户在编写程序时可直接使用所导入的命名空间的所有类和接口，而无须使用完全限定名来访问这些类和接口。

3. Assembly指令

Assembly 指令的功能是在编译期间将程序集关联到页面或用户控件上，使程序集的所有类和接口都可以在页面上使用。Assembly 指令支持 2 个属性，分别是 Name 和 Src。

（1）Name：允许指定用于关联页面文件的程序集名称。程序集名称应只包含文件名，不包含文件的扩展名，ASP.NET 引擎会按照系统路径逐一搜索，同时也会查找 Web 应用程序的

\Bin 目录。例如，文件是 MyAssembly.cs，Name 属性值应是 MyAssembly。

(2) Src: 允许指定编译时所使用的程序集源文件，需要指明源文件的全路径。例如，文件是 MyAssembly.cs，Src 属性值应是 MyAssembly.cs。

在同一个页面中，可以使用多条 Assembly 指令，但是在同一条 Assembly 指令中，Name 和 Src 属性只能任选其一。下面是使用 @Assembly 指令的 2 个例子：

```
<% @ Assembly Name=" MyAssembly"%>
<% @ Assembly Src=" MyAssembly.cs"%>
```

#### 4. Reference指令

Reference 指令的功能是将其他页面、用户控件或任何文件动态编译，并链接到当前页面。这样，用户就可以在当前文件内部引用这些对象及其公共成员。Reference 指令支持 3 个属性，分别是 Page、Control 和 VirtualPath。

(1) Page: 指定外部页，ASP.NET 应动态编译该页并将它链接到包含 @ Reference 指令的当前文件。

(2) Control: 指定外部用户控件，ASP.NET 应动态编译该控件并将它链接到包含 @ Reference 指令的当前文件。

(3) VirtualPath: 引用的虚拟路径，可以是任何文件类型。例如，它可能会指向母版页。下面是使用 @ Reference 指令的 3 个例子：

```
<% @ Reference Page =" MyPage.aspx"%>
<% @ Reference Control =" MyControl.ascx"%>
<% @ Reference VirtualPath =" MasterPage.master"%>
```

#### 5. Register指令

Register 指令的功能是将别名与命名空间和类名关联起来，作为定制服务器控件语法中的标记，这为开发人员提供了一种在 ASP.NET 应用程序文件中引用自定义控件的简明方法。把一个用户控件拖放到页面上，VS 2008 就会在页面上创建一个 Register 指令，这样就在页面上注册了用户控件，就可以通过特定名称在页面中访问该控件了。

Register 指令支持 5 个属性，分别是 assembly、namespace、src、tagname 和 tagprefix。

(1) assembly: 指定与 tagprefix 属性关联的命名空间所驻留的程序集。程序集名称不能包括文件扩展名。

(2) namespace: 指定正在注册的自定义控件的命名空间。

(3) src: 指定与 tagprefix:tagname 对关联的声明性 ASP.NET 用户控件文件的位置（相对的或绝对的）。

(4) tagname: 指定与类关联的任意别名。此属性只用于用户控件。

(5) tagprefix: 指定一个任意别名，它提供对包含指令的文件中所使用的标记的命名空间的短引用。

下面是使用 @ Register 指令把用户控件导入页面的 1 个例子：

```
<% @ Register tagprefix ="MyTag" tagname ="MyControl" Src =" MyControl.ascx" %>
```

6. Implements指令

Implements 指令的功能是允许 ASP.NET 实现特定的 .NET Framework 接口，如果页面需要实现多个接口，则可以使用多条 Implements 指令。Assembly 指令仅支持 interface 属性，指定要在页或用户控件中实现的接口。例如：

```
<%@ Implements interface =" System.Web.UI.IValidator" %>
```

7. 其他指令

除了上面介绍的指令外，还有如下的指令。

- (1) Control 指令：该指令与 Page 指令类似，用来定义 ASP.NET 页分析器和编译器使用的特定于用户控件 (.ascx 文件) 的属性。此指令只能用于 ASP.NET 用户控件（其源代码包包含在 .ascx 文件中）。
- (2) Master 指令：该指令定义 ASP.NET 页分析器和编译器使用的特定于母版页 (.master 文件) 的属性。
- (3) MasterType 指令：该指令提供一种方法，用于当通过 Master 属性访问 ASP.NET 母版页时，创建对该母版页的强类型引用。
- (4) PreviousPage 指令：该指令提供一种方法来获得上一页的强类型，可通过 PreviousPage 属性访问上一页。
- (5) OutputCache 指令：该指令以声明的方式控制 ASP.NET 页或页中包含的用户控件的输出缓存策略。

4.1.3 页生命周期

ASP.NET 页运行时，此页将经历一个生命周期，在生命周期中将执行一系列处理步骤。这些步骤包括初始化、实例化控件、还原和维护状态、运行事件处理程序代码及进行呈现。了解页生命周期非常重要，因为这样就能在生命周期的合适阶段编写代码，以达到预期效果。一般来说，页要经历表 4.2 概述的各个阶段。除了页生命周期阶段以外，在请求前后还存在应用程序阶段，但是这些阶段并不特定于页。表 4.2 列出了常规页的生命周期阶段。

表 4.2 常规页的生命周期阶段

阶 段	说 明
页请求	页请求发生在页生命周期开始之前。用户请求页时，ASP.NET 将确定是否需要分析和编译页（从而开始页的生命周期），或者是否可以在不运行页的情况下发送页的缓存版本以进行响应
开始	在开始阶段，将设置页属性，如 Request 和 Response。在此阶段，页还将确定请求是回发请求还是新请求，并设置 IsPostBack 属性。此外，在开始阶段期间，还将设置页的 UICulture 属性
页初始化	页初始化期间，可以使用页中的控件，并将设置每个控件的 UniqueID 属性。此外，任何主题都将应用于页。如果当前请求是回发请求，则回发数据尚未加载，并且控件属性值尚未还原为视图状态中的值
加载	加载期间，如果当前请求是回发请求，则将使用从视图状态和控件状态恢复的信息加载控件属性
验证	在验证期间，将调用所有验证程序控件的 Validate 方法，此方法将设置各个验证程序控件和页的 IsValid 属性

阶 段	说 明
回发事件处理	如果请求是回发请求，则将调用所有事件处理程序
呈现	在呈现之前，会针对该页和所有控件保存视图状态。在呈现阶段中，页会针对每个控件调用 <code>Render</code> 方法，它会提供一个文本编写器，用于将控件的输出写入页的 <code>Response</code> 属性的 <code>OutputStream</code> 中
卸载	完全呈现页并已将页发送至客户端、准备丢弃该页后，将调用卸载。此时，将卸载页属性（如 <code>Response</code> 和 <code>Request</code> ）并执行清理

4.2 ASP.NET内置对象

ASP.NET 定义了多个内置对象，它们是全局对象，即不必事先声明就可以直接使用。这些内置对象提供基本的请求、响应、会话等处理功能。ASP.NET 程序设计几乎不能没有对象，它是程序设计中使用的最频繁的元素之一。

4.2.1 Response对象

Response 对象用于向浏览器发送数据，数据以 HTML 的格式发送。Response 与 Request 对象组成了一对发送、接收数据的对象，这也是实现动态的基础。在 ASP.NET 中，Response 对象实际上是 System.Web 命名空间中的 `HttpResponse` 类，出于习惯仍然将 ASP.NET 中的 `HttpResponse` 类称为 Response 对象。

1. Response对象常用属性和方法

Response 对象有许多属性和方法，表 4.3 列出了 Response 对象的常用属性和方法。

表 4.3 Response 对象常用属性和方法

名 称	方法/属性	描 述
AddHeader	方法	将一个 HTTP 标头添加到输出流。提供 AddHeader 是为了与 ASP 的先前版本保持兼容
AppendHeader	方法	将 HTTP 头添加到输出流
AppendToLog	方法	将自定义日志信息添加到 Internet 信息服务（IIS）日志文件
BinaryWrite	方法	将一个二进制字符串写入 HTTP 输出流
Buffer	属性	获取或设置一个值，该值指示是否缓冲输出并在处理完整个响应之后发送它
BufferOutput	属性	获取或设置一个值，该值指示是否缓冲输出并在处理完整个响应之后发送它
Charset	属性	获取或设置输出流的 HTTP 字符集
Clear	方法	清除缓冲区流中的所有内容输出
ClearContent	方法	清除缓冲区流中的所有内容输出
ContentType	属性	获取或设置输出流的 HTTP MIME 类型
Cookies	属性	获取响应 Cookie 集合
End	方法	将当前所有缓冲的输出发送到客户端，停止该页的执行，并引发 EndRequest 事件
Flush	方法	向客户端发送当前所有缓冲的输出
IsClientConnected	属性	获取一个值，通过该值指示客户端是否仍连接在服务器上



名 称	方法/属性	描 述
Redirect	方法	将客户端重定向到新的 URL，并指定该新 URL
Write	方法	将信息写入 HTTP 响应输出流
WriteFile	方法	将指定的文件直接写入 HTTP 响应输出流

表中，Buffer 属性和 BufferOutput 属性功能相同，保留 Buffer 属性是为了与 ASP 兼容。Clear 方法与 ClearContent 方法功能相同，保留 Clear 方法也是为了与 ASP 兼容。

2. Response对象的应用

(1) 向浏览器发送信息

Response 对象最常用的方法是 Write，用于向浏览器发送信息。下面语句的功能是向浏览器输出“欢迎进入聊天室”的文本信息：

```
Response.Write("欢迎进入聊天室");
```

使用 Write 方法输出的字符串会被浏览器按 HTML 语法进行解释。因此可以使用 Write 方法直接输出 HTML 代码来实现页面内容和格式的定制。例如，下面的语句向浏览器输出红色的“欢迎进入聊天室”文本：

```
Response.Write("<font color=red>欢迎进入聊天室</font>"); //输出红色字体
```

(3) 重定向

Response 对象的 Redirect 方法可将当前网页导向指定页面，称为重定向。例如：

```
Response.Redirect("Page1.htm"); //将网页转移到当前目录的 Page1.htm
Response.Redirect("http://localhost/wh/who.htm"); //将网页转移到"/wh/who.htm"
```

(3) 缓冲处理

所谓缓冲处理，是指将输出暂时存放在服务器的缓冲区，待程序执行结束或接收到 Flush 或 End 指令后，再将输出数据发送到客户端浏览器。IIS 默认其为 True。Response 对象的 ClearContent（Clear）、Flush 和 ClearHeaders 三个方法用于缓冲处理。

例如，要将缓冲中的前一部分内容发送到浏览器，而后一部分内容删除，可以使用如下代码：

```
Response.BufferOutput = true; //启用缓冲
Response.Write("缓冲的前一部分，输出到浏览器");
Response.Flush(); //输出缓冲区内容
Response.Write("缓冲的后一部分，不输出到浏览器");
Response.ClearContent(); //清除缓冲区内容
```

(4) 结束程序运行

有时希望在某种条件下，提前结束网页运行并输出已生成的内容，这时可使用 Response 对象的 End 方法来实现。Response.End()方法的功能是结束程序的执行，若缓冲区有数据，则还会将其输出到客户端浏览器。End()方法的用法如下：

上面代码的功能是结束程序的执行，并将缓冲区内容输出到客户端浏览器。

4.2.2 Request对象

Request 对象派生自 HttpRequest 类，其主要功能是接收页面传递的信息，包括浏览器种类、用户输入表单中的数据、查询字符串、Cookies 中的数据和客户端认证等。

1. Request对象常用属性和方法

Request 对象有许多属性和方法，表 4.4 列出了 Request 对象的常用属性和方法。

表 4.4 Request 对象常用属性和方法

名 称	方法/属性	描 述
AcceptTypes	属性	获取客户端支持的 MIME 接收类型的字符串数组
Browser	属性	获取或设置有关正在请求的客户端的浏览器功能的信息
ClientCertificate	属性	获取当前请求的客户端安全证书
ContentEncoding	属性	获取或设置实体主体的字符串集
ContentLength	属性	指定客户端发送的内容长度（以字节计）
ContentType	属性	获取或设置传入请求的 MIME 内容类型
Cookies	属性	获取客户端发送的 Cookie 的集合
Files	属性	获取采用多部分 MIME 格式的由客户端上载的文件的集合
Form	属性	获取客户端表元素中所填入的信息
Headers	属性	获取 HTTP 头集合
Item	属性	从 Cookies、Form、QueryString 或 ServerVariables 集合中获取指定的对象
MapPath	方法	为当前请求将请求的 URL 中的虚拟路径映射到服务器上的物理路径
Params	属性	获取 QueryString、Form、ServerVariables 和 Cookies 项的数据
Path	属性	获取当前请求的虚拟路径
PathInfo	属性	获取具有 URL 扩展名的资源的附加路径信息
PhysicalApplicationPath	属性	获取当前正在执行的服务器应用程序的根目录的物理文件系统路径
QueryString	属性	获取 HTTP 查询字符串变量集合
RequestType	属性	获取或设置客户端使用的 HTTP 数据传输方法（GET 或 POST）
SaveAs	方法	将 HTTP 请求保存到磁盘
ServerVariables	属性	获取 Web 服务器变量的集合

2. 利用Request对象获取表单数据

获取表单数据是 Request 对象最主要的用途。动态网页最主要的特征是浏览器与服务器之间的交互性，客户端浏览器利用表单的提交将数据传送到服务器，服务器将检索结果回送浏览器。服务器获取表单数据的方式取决于客户端表单提交的方式。

（1）若表单的提交方式为“get”，则表单数据将以字符串形式附加在 URL 之后，在 QueryString 集合中返回服务器。例如：

上述中问号“?”之后即为表单中的项和数据值，表单项 XX 值为 value1，表单项 YY 值为 value2。

此时，在服务器端要使用 Request 对象的 QueryString 集合来获取表单数据。例如：

```
Request.QueryString["XX"];           // 获取表单项 XX 的值
Request.QueryString["YY"];           // 获取表单项 YY 的值
```

(2) 若表单的提交方式为“post”，则表单数据将放在浏览器请求的 HTTP 标头中返回服务器，其信息保存在 Request 对象的 Form 集合中。此时，在服务器端要使用 Request 对象的 Form 集合来获取表单数据。例如：

```
Request.Form["XX"];                   // 获取表单项 XX 的值
Request.Form["YY"];                   // 获取表单项 YY 的值
```

(3) 无论表单以何种方式提交，都可使用 Request 对象的 Params 集合来读取表单数据。例如：

```
Request.Params["XX"];                 // 获取表单项 XX 的值
Request.Params["YY"];                 // 获取表单项 YY 的值
```

或者，可以省略 QueryString、Form 或 Params，直接使用“Request[表单项]”来读取表单数据。例如：

```
Request["XX"];                        // 获取表单项 XX 的值
Request["YY"];                        // 获取表单项 YY 的值
```

使用 Params 集合或简略形式读取表单数据的处理过程是：Request 对象首先在 QueryString 集合中搜索表单项变量的值，若找到即返回相应值；否则，在 Form 集合中搜索，若找到也返回相应值；若都找不到，则返回 null。

**【例 4-2】** 设计一个用户登录页面，利用 Form 数据集合获取客户端提交的登录信息。设计步骤如下：

- ① 在 Chapter4 网站上添加一个命名为“example4-2”的单文件页模型网页。
- ② 切换到设计视图，从工具栏中拖放 2 个文本框（用于输入用户名和密码）和 1 个按钮（登录）控件，控件的 ID 属性值设置如图 4.2 所示。其中 ID 为 PWD 的文本框控件的 TextMode 属性设置为“Password”。

③ 双击按钮控件，系统自动切换到源视图并添加了 Button1\_Click 方法，在代码脚本块中添加如下代码（黑体代码）：

```
script runat="server">
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Write("你输入的用户名是： " + Request.Form["UserName"] + "<br>");
    Response.Write("你输入的密码是： " + Request.Form["PWD"] + "<br>");
}
</script>
```

④ 按【Ctrl+F5】组合键运行网页，当用户输入用户名和密码，并按下【登录】按钮以后，将显示用户输入的用户名和密码信息。程序的运行结果如图 4.3 所示。

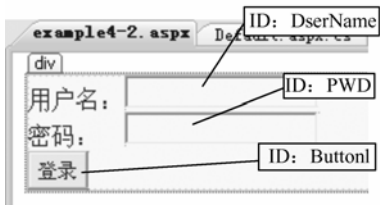


图 4.2 设置控件

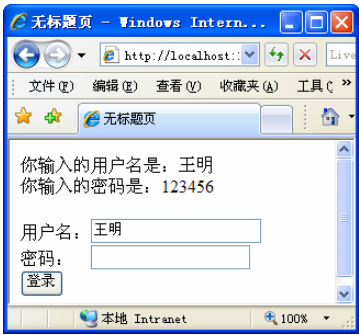


图 4.3 运行结果

### 3. 利用Request对象获取服务器端环境变量

Request 对象的 ServerVariables 数据集合可用来读取服务器端的环境变量信息。它由一些预定义的服务器环境变量组成，如发出请求的浏览器的信息、构成请求的 HTTP 方法、用户登录 Windows 的账号、客户端的 IP 地址等，这些变量为 ASP.NET 的处理带来了方便。这些变量都是只读变量。

下面的代码可以获取并输出 Web 服务器当前网页虚拟路径、当前网页实际路径、服务器主机名或 IP 地址、服务器连接端口、客户端主机名及浏览器信息：

```
Response.Write("当前网页虚拟路径:" + Request.ServerVariables["URL"]);
Response.Write("实际路径:" + Request.ServerVariables["PATH_TRANSLATED"]);
Response.Write("服务器名或 IP:" + Request.ServerVariables["SERVER_NAME"]);
Response.Write("服务器连接端口:" + Request.ServerVariables["SERVER_PORT"]);
Response.Write("客户主机名:" + Request.ServerVariables["REMOTE_HOST"]);
Response.Write("浏览器:" + Request.ServerVariables["HTTP_USER_AGENT"]);
```

### 4. 利用Request对象获取客户端浏览器能力信息

Request 对象的 Browser 数据集合是 HttpBrowserCapabilities 类型的对象，通过访问 Request 对象的 Browser 数据集合，可以容易地查询浏览器的能力。例如，下面的代码可以获取并输出浏览器描述、版本、是否支持 Cookie、是否支持 VBScript、DOM 版本号、是否安装 CLR、客户端操作系统等信息：

```
Response.Write("浏览器:" + Request.Browser.Browser);
Response.Write("版本:" + Request.Browser.Version);
Response.Write("支持 Cookie:" + Request.Browser.Cookies);
Response.Write("支持 VBScript:" + Request.Browser.VBScript);
Response.Write("微软 DOM 版本号:" + Request.Browser.MSDomVersion.ToString());
Response.Write("W3C DOM 版本号:" + Request.Browser.W3CDomVersion.ToString());
Response.Write("安装 CLR:" + Request.Browser.ClrVersion.ToString());
Response.Write("客户端操作系统:" + Request.Browser.Platform);
```

5. 利用Request对象获取客户端Cookie

Request 对象的 Cookies 数据集合用来记录客户端的信息，它由 HttpCookie 类派生，也常称其为 Cookie 对象。通常当浏览器访问 Web 服务器时，服务器使用 Response 对象的 Cookies 集合向客户端的 Cookie 写入信息，再通过 Request 对象的 Cookies 属性来检索 Cookie 信息。客户端 Cookie 存放于磁盘上，记录了浏览器的信息、何时访问 Web 服务器、访问过哪些页面等。Cookie 对象的属性如表 4.5 所示。

表 4.5 Cookie 对象的属性

名 称 及 值	描 述
Domain="..."	获取或设置可访问此 Cookie 对象的域
Expires="#Date#"	获取或设置 Cookie 的终止日期和时间
HasKeys	获取或设置 Cookie 对象中是否含有子键（Subkey）
Item[key]="val"	向 Cookie 中添加名为 key、值为 val 的子键；Item（key）可获得子键名为 key 的键值
Name="..."	获取或设置 Cookie 对象的名称
Value="..."	获取或设置 Cookie 对象的值
Values[key]="val"	向 Cookie 中添加名为 key、值为 val 的子键；Values（key）可获得子键名为 key 的键值

Cookie 对象的 Values 属性与 Item 属性功能相同，保留 Item 属性是为了与 ASP 兼容。

【例 4-3】 利用 Cookie 记录用户是否已投票，防止重复投票。当用户单击投票按钮时，向客户端写入一个生命周期为 10 天、名为“CookieExmp”的 Cookie 文件，其中包含三条记录，子键分别为“yaoming”、“kebi”和“Voted”，值分别为姚明的票数、科比的票数和当前时刻。

设计步骤如下：

- ① 在 Chapter4 网站上添加一个命名为“example4-3”的代码隐藏页模型网页。
- ② 切换到设计视图，从工具栏中拖放 2 个 Button（用于给姚明、科比投票）和 2 个 Label 控件（显示所投的票数）到页面上，如图 4.4 所示设置控件属性。
- ③ 分别双击两个 Button 控件，系统自动切换到代码页中并且分别添加了 Button1\_Click 方法和 Button2\_Click 方法。添加如下代码到 example4-3 局部类中（黑体为系统生成的代码）：

```
string hasvoted;  
protected void Page_Load(object sender, EventArgs e)  
{  
    HttpCookie cookie = Request.Cookies["CookieExmp"]; //通过 Request 的 Cookies 来检索  
                                                         Cookie 信息  
    if (cookie != null) //如果 cookie 不为空值，则读取  
                       cookie 的值  
    {  
        LblYaoming.Text = cookie.Values["yaoming"];  
        LblKebi.Text = cookie.Values["kebi"];  
        hasvoted = cookie.Values["Voted"];  
    }  
}
```

```

protected void Button1_Click(object sender, EventArgs e)
{
    if (hasvoted == null)                                //如果还未投票则调用 WriteCookies 方
                                                         法新建 Cookie
    {
        int score1, score2;
        score1 = System.Convert.ToInt32(LblYaoming.Text) + 1;
        score2 = System.Convert.ToInt32(LblKebi.Text);
        WriteCookies(score1, score2);
    }
    else
    {
        Response.Write("请不要重复投票! ");
    }
}

protected void Button2_Click(object sender, EventArgs e)
{
    if (hasvoted == null)                                //如果还未投票则调用 WriteCookies 方法
                                                         新建 Cookie
    {
        int score1, score2;
        score1 = System.Convert.ToInt32(LblYaoming.Text);
        score2 = System.Convert.ToInt32(LblKebi.Text) + 1;
        WriteCookies(score1, score2);
    }
    else
    {
        Response.Write("请不要重复投票! ");
    }
}

protected void WriteCookies(int YaomingScore, int KebiScore)
{
    HttpCookie MyCookie = new HttpCookie("CookieExmp");
    MyCookie["yaoming"] = YaomingScore.ToString();
    MyCookie["kebi"] = KebiScore.ToString();
    MyCookie["Voted"] = DateTime.Now.Date.ToShortDateString();
    MyCookie.Expires = DateTime.Today.AddDays(10d);
    Response.Cookies.Add(MyCookie);
    LblYaoming.Text = Request.Cookies["CookieExmp"]["yaoming"];
    LblKebi.Text = Request.Cookies["CookieExmp"]["kebi"];
}

```

④ 按【Ctrl+F5】组合键运行网页，程序运行后，当第一次单击【姚明】按钮时，姚明的得票将加 1，同时向客户端写入 Cookie，当再次单击【姚明】按钮时，读取 Cookie 并判断出用户已投过票，系统将提示“请不要重复投票！”，结果如图 4.5 所示。

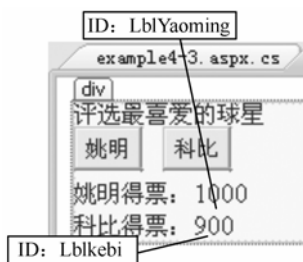


图 4.4 设置控件属性

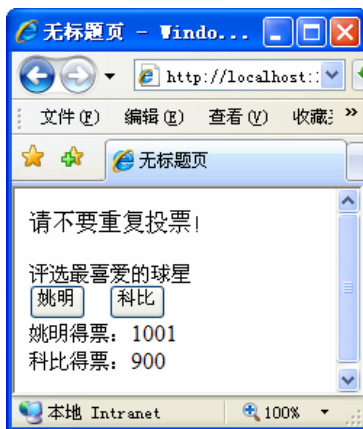


图 4.5 运行结果

## 4.2.3 Server对象

Server 是最基本的 ASP.NET 对象，派生自 `HttpServerUtility` 类，提供了服务器端的基本属性与方法。可通过 Page 对象的 `Server` 属性获取对应的 Server 对象，即 `Page.Server`，而通常 Page 可省略，直接使用 `Server` 进行操作。

### 1. Server对象常用属性和方法

Server 对象有许多属性和方法，表 4.6 列出了 Server 对象的常用属性和方法。

表 4.6 Server 对象的常用属性和方法

名 称	方法/属性	描 述
<code>ClearError</code>	方法	清除前一个异常
<code>CreateObject(type)</code>	方法	创建由 <code>type</code> 指定的对象或服务组件的实例
<code>Execute(path)</code>	方法	执行由 <code>path</code> 指定的 ASP.NET 程序，执行完毕后仍继续原程序的执行
<code>GetLastError()</code>	方法	获取最近一次发生的异常
<code>HtmlDecode</code>	方法	对已被编码以消除无效 HTML 字符的字符串进行解码
<code>HtmlEncode(string)</code>	方法	将 <code>string</code> 指定的字符串进行编码
<code>MachineName</code>	属性	服务器的计算机名称，为只读属性
<code>MapPath(path)</code>	方法	将参数 <code>path</code> 指定的虚拟路径转换成实际路径
<code>ScriptTimeout</code>	属性	获取或设置程序执行的最长时间，即程序必须在该段时间内执行完毕，否则将自动终止，时间以秒为单位。系统的默认值为 90 秒。例如， <code>ScriptTimeout = 100</code> ，表示最长程序执行时间为 100 秒
<code>Transfer(url)</code>	方法	结束当前 ASP.NET 程序，然后执行参数 <code>url</code> 指定的程序
<code>UrlDecode</code>	方法	对字符串进行解码，该字符串针对 HTTP 传输进行了编码并在 URL 中发送到服务器
<code>UrlEncode(string)</code>	方法	对 <code>string</code> 进行 URL 编码

## 2. 利用Server对象对HTML和URL进行编码和解码

Server 的 `HtmlEncode` 方法将对字符串进行编码，使它不被浏览器按 HTML 语法进行解释，而按字符串原样在浏览器中显示。`HtmlDecode` 方法的功能与 `HtmlEncode` 方法刚好相反，它可以对 HTML 编码的字符串进行解码。

**【例 4-4】** 在页面中直接显示 `example4-1.aspx` 程序源码。

设计步骤如下：

① 在 Chapter4 网站上添加一个命名为“example4-4”的代码隐藏页模型网页。

② 打开“`example4-4.aspx.cs`”代码页，添加命名空间“`using System.IO;`”，在 `Page_Load` 的方法体内添加如下代码：

```
// 新建 StreamReader 对象，打开 example4-1.aspx 文件，将内容读入到 tmp 字符串并输出
StreamReader reader = new StreamReader(File.Open(Server.MapPath("example4-1.aspx"),File
Mode.Open));
string tmp;
while ((tmp = reader.ReadLine()) != null)          //读取整行字符串赋给 tmp 并判断是否为空值
Response.Write(Server.HtmlEncode(tmp) + "<br>"); //tmp 输出页面并不被浏览器解释
reader.Close();
```

③ 按 **【Ctrl+F5】** 组合键运行网页，运行后的结果如图 4.6 所示。

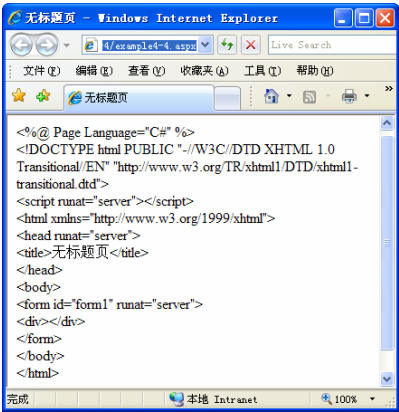


图 4.6 example4-4 页面运行的结果

## 3. 利用Server对象进行路径转换

程序中给出的文件路径通常使用的是虚拟路径，即相对于虚拟根目录的路径。例如，若虚拟目录 `xxx` 对应的实际路径为“`E:\TestASPNET\Test`”，则虚拟文件路径“`/abc.txt`”对应的实际路径为“`E:\TestASPNET\Test\abc.txt`”。有些应用中需要访问服务器的文件、文件夹或数据库文件，此时就需要将虚拟文件路径转换为实际文件路径。使用 `Server` 对象的 `MapPath` 方法可实现这种路径转换。例如：

```
Server.MapPath("/abc.txt")          // 返回文件 abc.txt 的实际路径名
Server.MapPath("/")                 // 返回虚拟根目录的实际路径名
```



4. 利用Server对象执行指定程序

Server 的 Execute 方法和 Transfer 方法都可让服务器执行指定的程序。Execute 类似于高级语言中的过程调用，将程序流程转移到指定的程序，当该程序执行结束后，流程将返回到原程序的中断点继续执行。而 Transfer 则是终止当前程序的执行，而转去执行指定的程序。例如：

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("<h3>输出本程序结果部分</h3><hr>");
        //页面上输出：输出本程序结果部分
        //执行 example4-3.aspx 页面
        Server.Execute("example4-3.aspx");
        //返回并在页面输出：输出 example4-3.aspx 的结果部分
        Response.Write("<hr><h3>输出 example4-3.aspx 的结果部分</h3>");
    }
</script>
```

4.2.4 Application对象

ASP.NET 应用程序是单个 Web 服务器上的某个虚拟目录及其子目录范围内的所有文件、页、处理程序、模块和代码的总和。Application 对象派生自 HttpSessionState 类，HttpSessionState 类的单个实例在客户端第一次从某个特定的 ASP.NET 应用程序虚拟目录中请求任何 URL 资源时创建。对于 Web 服务器上的每个 ASP.NET 应用程序都要创建一个单独的实例，然后通过内部 Application 对象公开对每个实例的引用。

Application 对象的一个常用的功能是存储应用程序级的全局变量，例如，可以将网站当前在线访问者的数量信息存储在 Application 对象中。

1. Application对象常用属性和方法

Application 对象有许多属性和方法，表 4.7 列出了 Application 对象的常用属性和方法。

表 4.7 Application 对象的常用属性和方法

名 称	方法/属性	描 述
Add(name,value)	方法	向 Contents 集合中添加名为 name、值为 value 的变量
AllKeys(index)	属性	只读。AllKeys 从 Content 集合中返回所有的变量名，AllKeys(index)返回下标为 index 的变量名
Clear	方法	清除 Contents 集合中的所有变量
Contents({name,index})	属性	从 Contents 集合中获取名为 name 或下标为 index 的变量值。例如 Application.Contents["cnt"]或简写为 Application["cnt"]。保留它是为了与 ASP 兼容
Count	属性	获取 Contents 集合中的变量数
Get({name,index})	方法	获取名为 name 或下标为 index 的变量值

名 称	方法/属性	描 述
GetKey(index)	方法	获取下标为 index 的变量名
Item({ name,index })	属性	从 Contents 集合内获取名为 name 或下标为 index 的变量值。例如 Application.Item["cnt"]或简写为 Application["cnt"]
Lock	方法	锁定对 HttpSessionState 变量的访问以促进访问同步,禁止其他用户修改 Application 对象的变量
Remove(name)	方法	从 Contentes 集合中删除名为 name 的变量
RemoveAll	方法	清除 Contents 集合中的所有变量
RemoveAt(index)	方法	删除 Contents 集合中下标为 index 的变量
Set(name,value)	方法	将名为 name 的变量值修改为 value
StaticObjects(name)	属性	获取 Global.asax 文件中的由 <object> 标记声明的所有对象
UnLock	方法	取消锁定对 HttpSessionState 变量的访问以促进访问同步,允许其他用户修改 Application 对象的变量

2. Application对象的事件

Application 事件处理程序只能在 Global.asax 文件中定义,且 Global.asax 文件必须存放在 Web 主目录中。当浏览器与 Web 服务器连接时,会先检查 Web 主目录中有没有 Global.asax 文件,如果有,则先执行该文件中定义的事件处理程序。

Application 对象有以下 4 个事件。

- (1) OnStart 事件: 在整个 ASP.NET 应用中首先被触发的事件,即在一个虚拟目录中第一个 ASP.NET 程序执行时触发。
- (2) OnEnd 事件: 与 OnStart 事件正好相反,在整个应用停止时被触发(通常发生在服务器被重启/关机时,或 IIS 被停止时)。
- (3) OnBeginRequest 事件: 在每一个 ASP.NET 程序被请求时就发生,即客户每访问一个 ASP.NET 程序时,就触发一次该事件。
- (4) OnEndRequest 事件: ASP.NET 程序结束时,触发该事件。

3. Application对象的应用

Application 对象主要用于访问同一网站的各个用户之间共享信息,记录整个网站的信息,如访问人数、在线人数、在线调查等。

**【例 4-5】** 使用 Application 对象的变量 cnt 对来访人数进行累计,即应用程序每执行一次,变量 cnt 即增加 1,同时在页面上显示来访人数的累计值。注意到,若有多个人同时访问网站,则对 cnt 变量的同时增 1 操作会造成最终 cnt 只被增 1,因此在进行 cnt 变量增 1 操作之前必须锁定 Application 对象,增 1 操作结束之后再开锁。

设计步骤如下:

- ① 在 Chapter4 网站上添加一个命名为“example4-5”的代码隐藏页模型网页和 Global.asax 文件。
- ② 在 Global.asax 文件的 Application\_Start 事件处理程序中创建计数器变量,代码如下:

```
protected void Application_Start(object sender, EventArgs e)
{
    Application.Set("cnt", 0); //创建访问计数器变量，初值为 0
}
```

③ 打开 example4-5.aspx.cs 代码页，在 Page\_Load 方法体内添加如下代码：

```
Application.Lock(); //锁定，不允许其他用户修改变量
Application.Set("cnt", (int)Application["cnt"] + 1); //访问计数增加 1
Application.UnLock(); //开锁，允许其他用户修改变量
Response.Write("您是第" + Application["cnt"] + "位来访者");
```

④ 按【Ctrl+F5】组合键运行网页，程序运行后刷新页面，数据在变化，如图 4.7 所示。  
Application["cnt"]变量可以像一般的变量一样进行操作，但与一般变量是有区别的。主要区别有两点。

- 生命周期不同：一般变量从程序开始执行时产生，程序执行结束时释放；而 Application 变量的生命始于 IIS 启动并有用户访问网页，它并不会因为程序执行结束而释放，只有在 Web 站点停止或者操作系统重启等情况下才被释放。
- 作用范围不同：一般变量对于每个访问者都有一个副本，而 Application 变量对所有应用程序只有一份，它相当于一个全局变量，“全局”指整个虚拟目录或站点下的应用程序。



图 4.7 example4-7 网页运行结果

## 4.2.5 Session对象

Session（会话）对象派生自 HttpSessionState 类，提供对会话状态值、会话级别设置和生存期管理方法的访问。它与 Application 对象一样，都是 ASP.NET 应用程序公用的对象。所不同的是，Application 对象是应用程序级别的公用对象，而 Session 是用户级别的公用对象。Session 对象用于在单个用户访问的各页面之间传递信息，即 Session 是同一链接所有网页的公用对象。例如，某个时刻有 10 位链接者，则 Session 对象个数为 10，每个链接者都有自己的 Session 对象，且互不相干，而 Application 对象个数为 1。

### 1. Session对象常用属性和方法

Session 对象有许多属性和方法，表 4.8 列出了 Session 对象的常用属性和方法。

表 4.8 Session 对象的常用属性和方法

名 称	方法/属性	描 述
Add(name,value)	方法	向 Contents 集合中添加名为 name、值为 value 的变量
Abandon	方法	释放 Session 对象，调用此方法将触发 OnEnd 事件
Clear	方法	清除 Contents 集合中的所有变量
CopyTo(array,index)	方法	复制 Session 对象的变量集合到 Array 指定的数组
Contents({name,index})	属性	从 Contents 集合中获取名为 name 或下标为 index 的变量值
Count	属性	获取 Contents 集合中的变量数
IsReadOnly	属性	只读。Session 是否为只读，默认为 False
IsNewSession	属性	只读。Session 对象是否与当前请求一起创建
Item({name,index})	属性	从 Contents 集合内获取名称为 name 或下标为 index 的变量值
Keys	属性	获取 Contents 集合内的所有变量。Keys(index)为获取下标为 index 的变量值
Mode	属性	获取当前会话状态模式
Remove(name)	方法	从 Contentes 集合中删除名为 name 的变量
RemoveAll	方法	清除 Contents 集合中的所有变量
RemoveAt(index)	方法	删除 Contents 集合中下标为 index 的变量
SessionID	属性	获取用于标识每个 Session 对象的标识码
StaticObjects	属性	获取由 ASP.NET 应用程序文件 Global.asax 中的 <object Runat="Server" Scope="Session"/> 标记声明的对象的集合
Timeout	属性	获取或设置 Session 对象的失效时间，单位为分钟，默认为 20 分钟

Session 对象的使用与 Application 对象非常相似，它也有两个集合（Contents 和 StaticObjects），用于存储变量和对象，变量与对象的设置与引用方法都与 Application 对象相同。例如，Session.Contents["cnt"]、Session.item["cnt"]和 Session["cnt"]，都表示访问 Session 对象的 cnt 变量。

2. Session对象事件

Session 对象有以下两个事件。

- （1）OnStart 事件：当用户第一次访问 ASP.NET 应用程序时将创建 Session 对象，并触发 OnStart 事件。对同一用户该事件只发生一次，除非发生 OnEnd 事件，否则不会再触发该事件。
- （2）OnEnd 事件：在 Timeout 属性所设置的时间内没有再访问网页，或者调用了 Abandon 方法都会触发此事件。该事件通常用于用户会话结束的处理，如将数据写入文件或数据库等。

**注意：**仅当会话状态 mode 被设置为 InProc 时，才会引发 OnEnd 事件。

3. 利用Session对象准确计数

在【例 4-5】中，无论用户是首次访问还是刷新网页，计数器都会加 1，这样就产生了重复计数。可以利用 Session 对象来区分是首次访问还是刷新网页，从而可以避免重复计数。打开 example4-5.aspx.cs 代码页，在 Page\_Load 方法体内改写代码，代码如下所示：

```

if(Session.IsNewSession)                                //判断 Session 与请求是否是同时创建的
{
    Application.Lock();                                  //锁定，不允许其他用户修改变量
    Application.Set("cnt", (int)Application["cnt"] + 1); //访问计数增加 1
    Application.Unlock();                                //开锁，允许其他用户修改变量
}
Response.Write("您是第" + Application["cnt"] + "位来访者");

```

程序运行后，当再次刷新时，计数器值不会加 1。

#### 4. 利用Session对象跨页传递参数

有时从一个页面转到另一个页面，同时需要将参数也传递到下一个页面供使用。利用 Session 对象，可以有效地进行跨页传参。

**【例 4-6】** 使用 Session 对象的变量来存储要传递的数据。在第一个页面中将数据保存到 Session 对象中，而在第二个页面中读取 Session 对象中的数据，从而实现数据的传递。

设计步骤如下：

① 在 Chapter4 网站上添加一个命名为“example4-6-a”和一个命名为“example4-6-b”的代码隐藏页模型网页。

② 打开 example4-6-a 页面的设计视图。从工具栏中拖放 2 个文本框（用于输入用户名和密码）和 1 个按钮（登录）控件，控件的 ID 属性值设置如图 4.8 所示。其中 ID 为 PWD 的文本框控件的 TextMode 属性设置为“Password”。双击按钮控件，系统自动切换到 example4-6-a.aspx.cs 代码页中并添加“Button1\_Click”方法。在 Button1\_Click 方法体中添加如下代码：

```

Session["UName"] = UserName.Text;                        //保存 Session 变量 UName
Session["Pass"] = PWD.Text;                              //保存 Session 变量 Pass
Response.Redirect("example4-6-b.aspx");                  //重定向到 example4-6-b.aspx

```

③ 打开 example4-6-b.aspx.cs 代码页，在 Page\_Load 方法体中添加如下代码：

```

Response.Write("你输入的用户名是：" + Session["UName"] + "<br>");
Response.Write("你输入的密码是：" + Session["Pass"] + "<br>");

```

④ 打开 example4-6-a.aspx 页面，按【Ctrl+F5】组合键运行网页，输入用户名“admin”、密码“123456”，单击【登录】按钮跳转到 example4-6-b.aspx 页面，结果如图 4.9 所示。

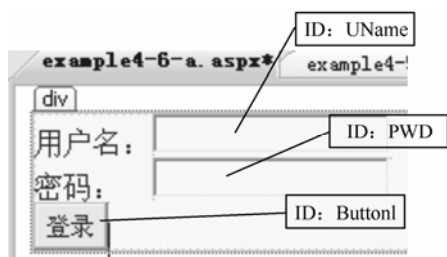


图 4.8 设置控件属性



图 4.9 example4-6-b 运行结果

## 5. 利用Session对象强制登录

很多网站对没有登录的用户的访问是有限制的，对有些网页只允许已登录的用户访问，为了防止未登录用户跳过登录页而直接访问受保护的页面，可以利用 Session 对象来判断用户的登录状态，若未登录则将其请求转到登录页面，从而起到了保护资源的目的。

例如，对【例 4-6】中的代码重新改写。打开 example4-6-a.aspx.cs 代码页，在 Button1\_Click 方法体中重新输入以下代码：

```
Session["UName"] = UserName.Text;           //保存 Session 变量 UName
Session["Pass"] = PWD.Text;                 //保存 Session 变量 Pass
Session["Logged"] = true;
Response.Redirect("example4-6-b.aspx");      //重定向到 example4-6-b.aspx
```

打开 example4-6-b.aspx.cs 代码页，在 Page\_Load 方法体中重新输入以下代码：

```
if (Session["Logged"] == null)
{ Response.Redirect("example4-6-a.aspx"); }
Response.Write("你输入的用户名是： " + Session["UName"] + "<br>");
Response.Write("你输入的密码是： " + Session["Pass"] + "<br>");
```

运行 example4-6-b.aspx 页面，因为没有登录，Session["Logged"] 值为 null，所以被强制转到 example4-6-a.aspx 页面。

### 4.2.6 Page 对象

当浏览器打开 Web 网页时，ASP.NET 先编译 Web 网页，分析网页及其代码，然后以动态的方式产生新的类，再编译新的类。Web 网页编译后所创建的类由 Page 类派生而来，因此，Web 网页可以使用 Page 类的属性、方法与事件。

每次请求 Web 网页，新派生的类将成为一个能在服务器执行的可执行文件。在运行阶段，Page 类会以动态的方式创建 HTML 标记并返回浏览器，同时处理收到的请求（Request）和响应（Response）。若网页中包含服务器控件，则 Page 类可作为服务器控件的容器，并在运行阶段创建服务器控件。

#### 1. Page 对象常用属性和方法

Page 对象有许多属性和方法，表 4.9 列出了 Page 对象的常用属性和方法。

IsPostBack 是 Page 对象的一个重要属性。这是一个只读的 Boolean 类型属性，指示页面是第一次加载还是为了响应客户端回传而进行的加载。有经验的程序员将一些耗费资源的操作（例如，从数据库获取数据或构造列表项）放在页面第一次加载时执行。如果页面回传到服务器并再次加载，就无须重复这些操作了。

#### 2. Page 对象事件

Page 对象最主要的事件是 Init、Load 和 UnLoad 事件，在 Init 和 Load 事件中都可以进行页面的初始化操作。虽然 Init 和 Load 事件在网页加载时都会被触发，但它们是有区别的。

表 4.9 Page 对象的常用属性和方法

名 称	方法/属性	描 述
Application	属性	获取目前 Web 请求的 Application 对象，Application 对象派生自 httpapplicationstate 类，每个 Web 应用程序都有一个专属的 Application 对象
Cache	属性	获取与网页所在应用程序相关联的 Cache 对象，Cache 对象派生自 Cache 类，允许在后续的请求中保存并捕获任意数据，Cache 对象主要用来提升应用程序的效率
ClientScript	属性	获取用于管理脚本、注册脚本和向页添加脚本的 ClientScriptManager 对象
ClientTarget	属性	获取或设定数值，覆盖浏览器的自动侦测，并指定网页在特定浏览器用户端如何显示。若设置了此属性，则会禁用客户端浏览器检测，使用在应用程序配置文件（web.config）中预先定义的浏览器能力
Controls	属性	获取 ControlCollection 对象，该对象表示 UI 层次结构中指定服务器控件的子控件
DataBind	方法	将数据源绑定到网页上的服务器控件
EnableViewState	属性	获取或设置目前网页请求结束时，网页是否要保持视图状态及其所包含的任何服务器控件的视图状态（viewstate），默认为 true
ErrorPage	属性	获取或设置当网页发生未处理的异常情况时，要将用户定向到哪个错误信息网页，此属性可以让用户自定义所要显示的错误信息。如果没有设置此属性，则 ASP.NET 会显示默认的错误信息网页
FindControl	方法	在网页上搜索标志名称为 id 的控件，返回值为标志名称为 id 的控件，若找不到标志名称为 id 的控件，则会返回 Nothing
HasControls	方法	获取布尔值，用来判断 Page 对象是否包含控件，返回 True 表示包含控件，返回 False 表示没有包含控件
IsPostBack	属性	获取布尔值，用来判断网页在何种情况下加载，返回 False 表示是第一次加载该网页，返回 True 表示是因为客户端返回数据而被重新加载
IsValid	属性	获取布尔值，用来判断网页上的验证控件是否全部验证成功，返回 True 表示全部验证成功，返回 False 表示至少有一个验证控件验证失败
MapPath	方法	将 VirtualPath 指定的虚拟路径（相对或绝对路径）转换成实际路径
Request	属性	获取请求网页的 Request 对象，Request 对象派生自 HttpRequest 类，主要用来获取客户端的相关信息
Response	属性	获取与请求网页相关的 Response 对象，Response 对象派生自 HttpResponse 类，允许发送 HTTP 响应数据给客户端
Server	属性	获取 Server 对象，Server 对象派生自 HttpServerUtility 类
Session	属性	获取 Session 对象，Session 对象派生自 HttpSessionstate 类
Trace	属性	获取目前 Web 请求的 Trace 对象，Trace 对象派生自 TraceContext 类，可以用来处理应用程序跟踪
Validators	属性	获取请求的网页所包含的 ValidatorsCollection（验证控件集合），网页上的验证控件均存放在此集合中

(1) Init 事件

当一个用户多次请求同一个网页（Page）时，Init 事件在每一次请求时被触发。由于视图状态尚未加载，所以在该事件的生存期内不应访问其他服务器控件。

### (2) Load 事件

当一个用户多次请求同一个网页 (Page) 时, Load 事件在每一次请求时被触发, 可以在此事件中访问控件。可以使用 Page 对象的 IsPostBack 属性来判断是否是第一次请求。若为 False 则是第一次请求, 否则为回发。

### 3. 利用Page对象进行页面的初始化

ASP.NET 是事件驱动的应用程序, Page\_Load 事件是当一个页面开始执行时, 只要定义了 Page\_Load 事件的处理方法, 就会最先执行该处理方法。通常在这个事件处理程序中进行页面的初始化。

**【例 4-7】** 设计一个页面, 显示一个选择论坛主题的下拉列表, 当该页面被请求时, 在 Page\_Load 事件中动态初始化列表项内容。

设计步骤如下:

- ① 在 Chapter4 网站上添加一个命名为 “example4-7” 的代码隐藏页模型网页。
- ② 打开 example4-7 页面的设计视图。从工具栏中拖放 1 个下拉列表框 DropDownList (用于显示论坛主题) 控件, 如图 4.10 所示。
- ③ 打开 example4-7.aspx.cs 代码页, 在 Page\_Load 方法体中添加如下代码:

```
if (!Page.IsPostBack)
{
    DropDownList1.Items.Add("课程");
    DropDownList1.Items.Add("考试");
    DropDownList1.Items.Add("综合");
}
```

④ 按【Ctrl+F5】组合键运行网页, 运行结果如图 4.11 所示。当用户首次请求 example4-7.aspx 页面时, 触发 Page\_Load 事件, 程序为下拉列表框初始化, 添加 3 个选项。如果用户再次刷新网页, 虽然也会触发 Page\_Load 事件, 但此时 Page.IsPostBack 值为 true, 因此不会为下拉列表框再添加 3 个选项。



图 4.10 页面设计图



图 4.11 example4-7 网页运行的结果

### 4.2.7 Cache对象

在 ASP.NET 中, Cache 对象实际上是 System.Web 命名空间中的 HttpCachePolicy 类, 出于习惯, 仍然将 ASP.NET 中的 HttpCachePolicy 类称为 Cache 对象。Cache 对象用于设置



ASP.NET 应用程序的缓存，有关缓存的内容将在后面的章节中详细介绍。

## 习 题

1. 是否要先创建内置对象的实例，然后才能使用？
2. Request、Response 对象的主要功能是什么？
3. 获取服务器的名称，可以利用（     ）对象。  
A. Response                      B. Session                      C. Server                      D. Cookie
4. Application 对象的特点包括（     ）。  
A. 数据可以在 Application 对象内部共享  
B. 一个 Application 对象包含事件，可以触发某些 Application 对象脚本  
C. 个别 Application 对象可以用 IIS 来设置而获得不同的属性  
D. 单独的 Application 对象可以隔离出来，在它们自己的内存中运行
5. 试述 Session 对象的生命周期。
6. Web 应用程序启动或者终止时，将激发\_\_\_\_\_和\_\_\_\_\_全局事件。
7. 实现两个页面，利用 Response 对象和 Request 对象在页面间进行内容传递。
8. 实现两个页面，利用 Session 对象进行内容传递。

# 第 5 章 ASP.NET 服务器控件和客户端脚本

通常，ASP.NET 应用程序由界面部分和用户接口逻辑程序两部分组成，设计应用程序界面需要使用各种标准控件。这些控件是 Web 页面能够容纳的对象之一，是构建 ASP.NET 应用程序的基础。本章将介绍服务器控件的使用，如何在 ASP.NET 页面中使用 JavaScript 改变服务器控件的操作，以及如何如何进行客户端回调。

## 5.1 控件概述

控件是一种类，绝大多数控件都具有可视的界面，能够在程序运行中显示其外观。利用控件进行可视化设计既直观又方便，可以实现所见即所得的效果。程序设计的主要内容是选择和设置控件，以及对控件的事件编写处理代码。

服务器控件是指在服务器上执行程序逻辑的组件，通常具有一定的用户界面，但可能不包括用户界面。服务器控件包含在 ASP.NET 页面中，在运行页面时，用户可与控件发生交互行为。当页面被用户提交时，控件可在服务器端引发事件，服务器端则根据相关事件处理程序来进行事件处理。服务器控件是动态网页技术的一大进步，它真正地将后台程序和前端网页融合在一起。

ASP.NET 提供了两种不同类型的服务器控件：HTML 服务器控件和 Web 服务器控件。这两种控件迥然不同：HTML 服务器控件会映射为特定的 HTML 元素，而 Web 服务器控件映射为 ASP.NET 页面上需要的特定功能。System.Web.UI.HtmlControls 名称空间包含 HTML 服务器控件，而 System.Web.UI.WebControls 名称空间包含 Web 服务器控件。ASP.NET 网页控件的层次结构如图 5.1 所示。

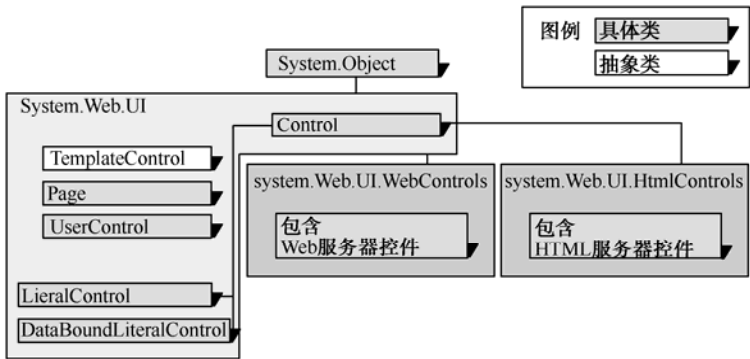


图 5.1 ASP.NET 网页控件的层次结构

ASP.NET 还提供一系列的验证服务器控件，利用这些控件可以方便地完成页面的数据验证，从而确保用户在应用程序的窗体中输入信息的有效性。ASP.NET 允许开发人员创建自己的用户控件，可以在设计视图中编辑它，再将它嵌入到其他 ASP.NET 网页中，然后将它们集

成进 ASP.NET 应用程序。开发人员还可以自定义服务器控件，将它添加进 VS 2008 的工具箱，在开发应用程序时，可以像拖曳其他 Web 标准控件那样方便地使用它。

## 5.2 HTML服务器控件

HTML 服务器控件运行在服务器上，并且可以直接映射为大多数浏览器支持的标准 HTML 标签。HTML 服务器控件由普通 HTML 控件转换而来，其外观基本上与普通 HTML 控件一致。

默认情况下，服务器无法使用 Web 窗体页上的 HTML 元素，这些元素被视为传递给浏览器的不透明文本。将 HTML 元素转换为 HTML 服务器控件，可将其公开为在服务器上可编程的元素。ASP.NET 允许提取 HTML 元素，通过少量的工作，把它们转换为服务器端控件。在源视图中对 HTML 元素添加 `runat="server"` 属性，即可将 HTML 元素转换为服务器控件。另外，为了让控件在服务器端代码中被识别出来，还应当添加 `id` 属性。

### 5.2.1 HTML服务器控件的层次结构

HTML 服务器控件位于名称空间 `System.Web.UI.HtmlControls` 中。图 5.2 显示了 HTML 服务器控件的层次结构。

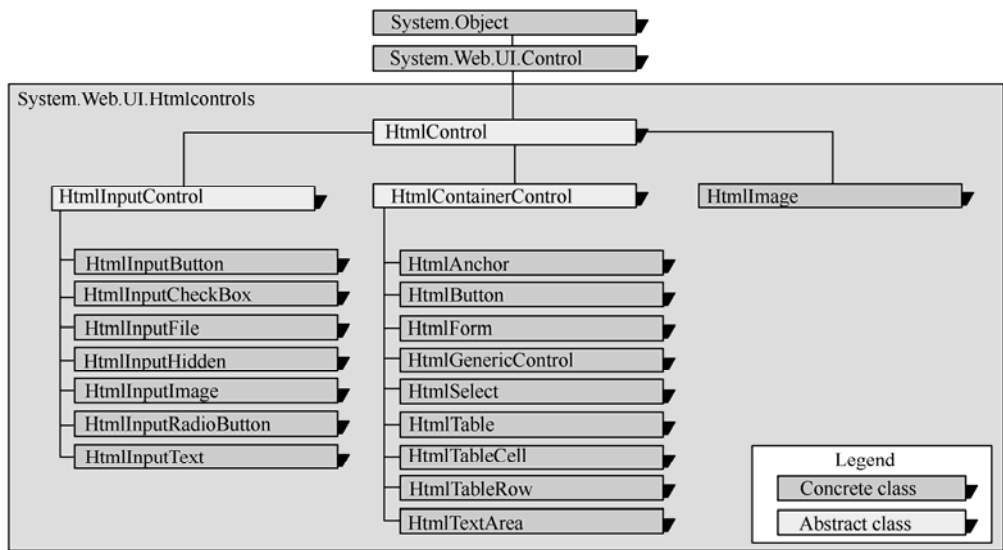


图 5.2 HTML 服务器控件的层次结构

### 5.2.2 HTML服务器控件的基本语法

定义 HTML 服务器控件的基本语法格式如下：

```
<HTML 标记 Id="控件名称" Runat="Server">
```

HTML 服务器控件是由 HTML 标记所衍生出来的新功能，在所有的 HTML 服务器控件的语法中，最前端是 HTML 标记，不同控件所用的标记不同；`Runat = "Server"` 表示控件将会在服务器端执行；`Id` 用来设置控件的名称，在同一程序中各控件的 `Id` 均不相同，`Id` 属性允许

以编程方式引用该控件。

表 5.1 列举了 HTML 服务器控件与 HTML 标签的对应表示，并标示出了它们所属的类别。

表 5.1 HTML 服务器控件与 HTML 标签的对应

HTML 服务器控件名称	类 别	HTML 标签	说 明
HtmlHead	容器	<head>	<head>元素，可以在其控件集合中添加其他元素
HtmlTitle	容器	<title>	标题元素
HtmlForm	容器	<form>	每个页面至多有一个 HtmlForm 控件
HtmlInputButton HtmlInputSubmit HtmlInputReset HtmlInputCheckbox HtmlInputFile HtmlInputHidden HtmlInputImage HtmlInputRadioButton HtmlInputText HtmlInputPassword	输入	<input>	<input type=button> <input type=submit> <input type=reset> <input type=checkbox> <input type=file> <input type=hidden> <input type=image> <input type=radio> <input type=text> <input type=password>
HtmlImage	空	<img>	图片
HtmlLink	空	<link>	读取/设置目标 URL
HtmlTextArea	容器	<textarea>	多行文本输入框
HtmlAnchor	容器	<a>	锚标签
HtmlButton	容器	<button>	服务器端按钮，可自定义显示格式，IE 4.0 及以上版本可用
HtmlMeta	容器	<meta>	<meta> 元素是关于呈现页的数据（但不是页内容本身）的容器
HtmlTable	容器	<table>	表格，可以包含行，行中包含单元格
HtmlTableCell	容器	<td>/<th>	表格单元格/表格标题单元格
HtmlTableRow	容器	<tr>	表格行，行中包含单元格
HtmlSelect	容器	<select>	用于选择的下拉菜单
HtmlGenericControl	容器	<span>、<div>、 <body>、<font>	此类可以表示不直接用 .NET Framework 类表示的 HTML 服务器控件元素

5.2.3 HTML服务器控件的应用

服务器不会处理普通的 HTML 控件，如<h1>、<a>和<input>，它们将直接被发送到客户端，由浏览器进行显示。如果要想 HTML 控件能在服务器端被处理，就要将它们转换为 HTML 服务器控件。

将普通 HTML 控件转换为 HTML 服务器控件，只需简单地添加 runat="server"属性。另外，可能还需要添加 id 属性，这样就可以通过编程方式访问和控制控件了。例如，下面是一个简单的输入控件：

```
<input type="text" size="40">
```

可以添加 id 和 runat 属性，将它转换为 HTML 服务器控件，如下所示：

```
<input type="text" id="BookTitle" size="40" runat="server">
```

因为 HTML 控件只能运行在客户端，而不是服务器端，所以往往在 ASP.NET 下运行现有的 HTML 页时需进行上述转换。转换后的 HTML 控件具有服务器控件的特点：可以使用面向对象技术对其进行编程控制；提供了一组事件，可以为事件编写事件处理程序；自动维护控件，允许自定义属性等。

在表 5.1 所列出的控件中，有几个共同的属性会经常被使用。这几个属性分别是 InnerHtml、InnerText、Disabled、Visible、Value、Attributes 及 Style。

### 1. InnerHtml 属性

获取或设置控件的开始标记和结束标记之间的内容，但不自动将特殊字符转换为等效的 HTML 实体。

### 2. InnerText 属性

获取或设置控件的开始标记和结束标记之间的内容，并自动将特殊字符转换为等效的 HTML 实体。

### 3. Value 属性

该属性用来获取各种输入字段的值，包括 HtmlSelect、HtmlInputText 等。

### 4. Attributes 属性

该属性是服务器控件标记上表示的所有属性名称和值的集合。使用该属性可以用编程方式访问 HTML 服务器控件的所有特性。

**【例 5-1】** 使用 HTML 服务器控件实现一个简单的提交表单。在服务器端编程获取用户所填写的内容并输出。

(1) 运行 VS 2008，新建一个网站，并将其命名为“Chapter5”。

(2) 右击“解决方案资源管理器”内的项目名称，选择“添加新项”选项，再选择新建“Web 窗体”，并输入文件名“example5-1”，单击【确定】按钮。

切换到“example5-1”的设计视图，单击菜单项“表”→“插入表”，在出现的窗口中把表格设置为 7 行 2 列。选择最后一行单元格→右击所选择的最后一行→合并单元格，切换到拆分视图，并添加此单元格 runat="server" 属性，将其 id 设为 result，如下所示：

```
<td colspan="2" style="height: 81px; text-align: left" id="result" runat="server"></td>
```

(3) 在表格的第 1 列输入如图 5.4 所示的文本。

(4) 打开工具栏，展开 HTML 工具箱，如图 5.3 所示，拖放对应的控件，如图 5.4 所示。

① 切换到拆分视图，在第 1 行拖放一个 input (Text) 控件，添加 runat="server" 属性，如下所示：

```
<input id="name" type="text" runat="server"/> // runat="server"使其成为 HTML 服务器控件
```

② 在第 2 行拖放两个 input (Radio)，并输入“男”和“女”，添加如下所示的属性：

```
<input id="male" name="sex" type="radio" value="男" runat="server" />男  
<input id="female" name="sex" type="radio" value="女" runat="server" />女
```

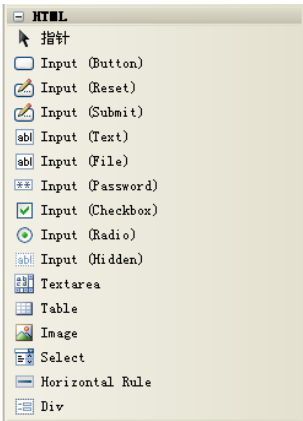


图 5.3 HTML 工具箱

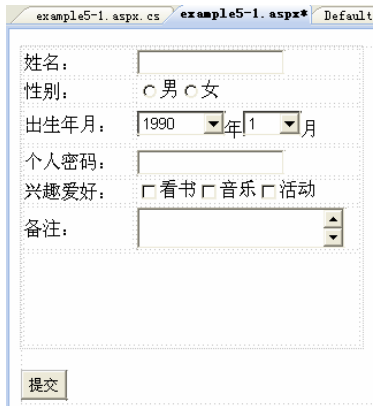


图 5.4 example5-1 页面设计

③ 在第 3 行拖放两个 Select 控件，并在其后输入“年”和“月”。添加如下所示的属性：

```
<select id="year" name="D1" runat="server" ><option>1990</option><option>1991</option>  
<option>1992</option><option>1993</option><option>1994</option><option>1995</option>  
</select>年  
  
<select id="month" name="D2" runat="server"><option>1</option><option>2</option>  
<option>3</option><option>4</option><option>5</option><option>6</option><option>7</option>  
<option>8</option><option>9</option><option>10</option><option>11</option><option>12  
</option>  
</select>月</td>
```

④ 在第 4 行拖放一个 input (Password) 控件，添加如下所示的属性：

```
<input id="pwd" type="password" runat="server" />
```

⑤ 在第 5 行拖放 3 个 input (checkbox) 控件，分别在控件后输入“看书”、“音乐”和“活动”，并分别设置 id 为“book”、“music”和“sport”，其属性如下所示：

```
<input id="book" type="checkbox" runat="server"/>看书<input id="music" type="checkbox"  
runat="server"/>音乐<input id="sport" type="checkbox" runat="server" />活动
```

⑥ 在第 6 行拖放一个 Textarea 控件，并设置其 id 为“remark”。属性如下所示：

```
<textarea id="remark" name="S1" runat="server"></textarea>
```

⑦ 在表格下面拖放一个 input (button) 控件，并设置其为 HTML 服务器控件，value 属性值设置为“提交”：

```
<input id="Button1" type="button" value="提交" runat="server" onserverclick="Button1_Click" />
```

(5) 打开 example5-1.aspx.cs 代码页，添加事件处理程序，输入以下代码：

```
protected void Button1_ServerClick(object sender, EventArgs e)
{
    string strHtml = "姓名: " + name.Value + "<br/>";
    strHtml += "性别: " + (male.Checked ? "男<br/>" : "女<br/>");
    strHtml += "出生年月: " + year.Value + "年" + month.Value + "月<br/>";
    strHtml += "个人密码: " + pwd.Value + "<br/>";
    strHtml += "兴趣爱好: " + (book.Checked ? book.Value : " ") + (music.Checked ?
        music.Value : " ") + (sport.Checked ? sport.Value : " ") + "<br/>";
    strHtml += "备注: " + remark.Value + "<br/>";
    result.InnerHtml = strHtml;
}
```

(6) 按【Ctrl+F5】组合键运行网页，在表单内输入相关内容，单击【提交】按钮，运行结果如图 5.5 所示。

姓名: 王明  
 性别: ☒ 男 ☐ 女  
 出生年月: 1991 年 6 月  
 个人密码: 123456  
 兴趣爱好: ☒ 看书 ☒ 音乐 ☐ 活动  
 备注: 三好学生  
 提交

图 5.5 运行结果

如果查看 default.aspx 文件源视图中按钮的 HTML 代码，则可以发现 Button1 按钮包含一个 onServerClick 属性，而不是常规 HTML 或 ASP 页面中使用的 onClick 属性。这就在告知服务器，当按钮的 Click 事件发生时，应调用的函数是“Button1\_ServerClick”。

如果希望控制在客户端处理事件，那么应使用传统的 onClick 属性。在这种情况下，必须提供客户端脚本来处理事件，系统会首先执行客户端代码，然后再运行服务器端代码。

## 5.3 Web服务器控件

在 VS 2008 的工具箱中，只有 HTML 选项卡中的控件是浏览器端控件，其他各种控件都是服务器控件。所有呈现到浏览器的具有可视化外观的 Web 服务器控件，都从 WebControl 类派生。

### 5.3.1 Web服务器控件的层次结构

从 Control 类派生的 WebControl 类和控件，位于 System.Web.UI.WebControls 命名空间中。其中包含表单控件（输入与显示控件、按钮控件、超链、日历控件、图像等）、列表控件、数

据源控件、数据绑定与数据显示控件、验证控件等。它们之间的关系如图 5.6 所示。

服务器控件包含方法及与之关联的事件处理程序，并且这些代码都在服务器端执行。部分服务器控件也提供客户端脚本，尽管如此，这些控件事件仍然会在服务器端处理。

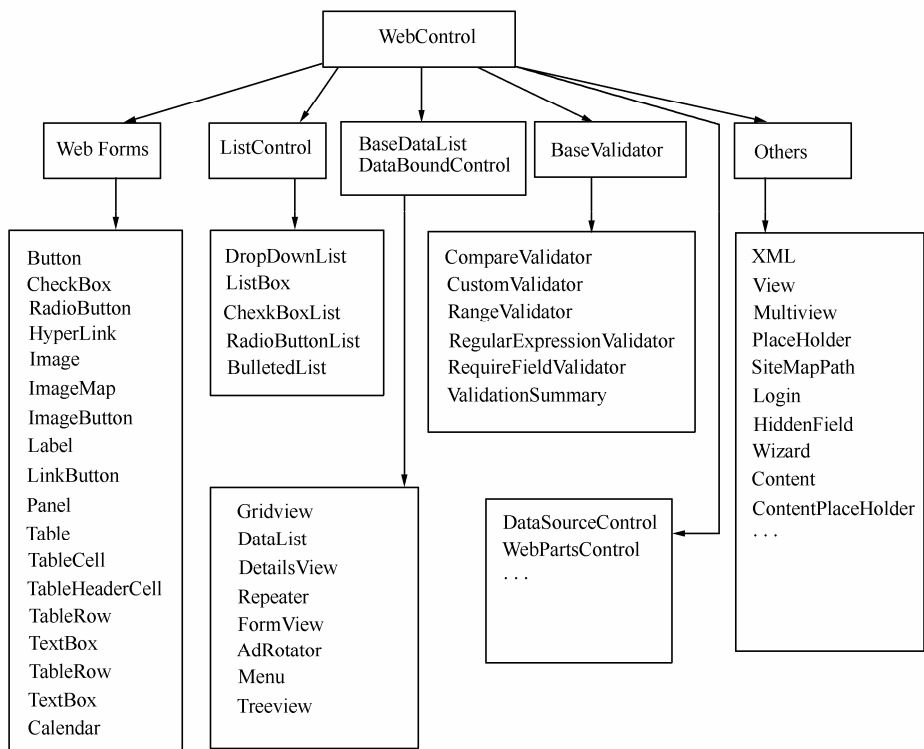


图 5.6 Web 服务器控件的层次结构

### 5.3.2 Web服务器控件的基本语法

ASP.NET 服务器控件的基本语法格式如下：

```
<asp:controlType id="ControlID" runat="server" thisProperty="this value" thatProperty="that value"/>
```

控件标签以 `asp:` 开头，这就是标记前缀。`controlType` 是控件的类型或类，如 `Button`、`CheckBoxList`、`GridView` 等。可以利用 `id` 属性，以编程方式引用控件实例。`runat` 属性告知服务器，该控件在服务器端运行。

可以在尖括号中声明其他属性。例如，为 `TextBox` 声明 `Text` 和 `Width` 属性，如下所示：

```
<asp:TextBox ID="txtBookName" runat="server" Width="250px" Text="Enter a book name.">
</asp:TextBox>
```

虽然 ASP.NET 允许标记的属性值可以不加引号，但是鉴于 ASP.NET 服务器控件必须使用良构的 XHTML 语法，所以不建议大家这样做。需要明确的是，如上所示，一般情况下，标签是成对出现的，也就是由起始标签 `<asp:TextBox>` 和结束标签 `</asp:TextBox>` 构成。但若此标签仅占一行，则也可在标签最后加一个 `/` 作为结束。所以，上面的 `TextBox` 也可以写为：



```
<asp:TextBox ID="txtBookName" runat="server" Width="250px" Text="Enter a book name."/>
```

另外，许多 Web 服务器控件可以在起始标签和结束标签之间使用内部 HTML。例如，在 TextBox 控件中，可将 Text 属性指定为内部 HTML，而不是将其设置在打开标签的属性中。所以，上面的控件又可以等价地写为：

```
<asp:TextBox ID="txtBookName" runat="server" Width="250px">Enter a book name.
</asp: TextBox>
```

5.3.3 Web服务器控件的属性

Web 服务器控件继承了 WebControl 和 System.Web.UI.Control 类的所有属性、事件和方法。表 5.2 列出了从 Control 或 WebControl 类继承的 Web 服务器控件的常用属性。

表 5.2 Web 服务器控件的常用属性

名 称	类 型	值	说 明
AccessKey	String	单字符的字符串	定义控件的加速键。例如，定义某控件的“AccessKey”属性值为“W”，就可以通过按【Alt+W】组合键来访问该控件
BackColor	Color	Azure、Green、Blue 等	背景颜色
BorderColor	Color	Fuchsia、Aqua、Coral 等	边框颜色
BorderStyle	BorderStyle	Dashed、Dotted、Double、NotSet 等	边框样式。默认为 NotSet
BorderWidth	Unit	nn、nnpt	边框的宽度。如果用 nn，则 nn 是整数，单位是像素。如果用 nnpt，则 nn 是整数，单位是点
Controls	ControlCollection		该控件所包含的所有控件对象的集合
CssClass	String		CSS 类
Enabled	Boolean	true、false	若设为 false，则控件可见，但显示为灰色，不能操作，内容仍可复制和粘贴。默认值为 true
Font	FontInfo		定义控件上显示的文本的格式
ForeColor	Color	Lavender、LightBlue、Blue 等	前景色
ID	String		控件的可编程标识符
Parent	Control	页面上的控件	返回在页面控件层次结构中对该控件的父控件的引用
SkinID	String	皮肤文件名	应用到该控件的主题目录下的皮肤文件的详细信息
ToolTip	String		当鼠标移动到控件上方的时候显示的文本字符串，在低版本的浏览器中呈现
Visible	Boolean	true、false	若设为 false，则不呈现该控件。默认值为 true

一般可以使用服务器控件类的属性，来设置 ASP.NET 服务器控件的属性标记，并通过编程方式访问它们。而一旦控件被声明，或在代码中被实例化，就可以通过编程方式获取或设置其属性。

5.3.4 Web服务器控件的事件

Web 页面和控件都包含事件，它们继承自 **Control** 类（在 **Error** 事件的情况下，则继承自 **TemplateControl** 类）。所有这些事件都传递没有属性的 **EventArgs** 类型的事件参数。表 5.3 中列举了常见的控件事件。

表 5.3 常见的控件事件

事件名称	说明
<b>DataBinding</b>	当控件绑定到数据源时发生
<b>Disposed</b>	当控件从内存中释放时发生
<b>Error</b>	只在页面中，当抛出未处理的异常时发生
<b>Init</b>	当控件初始化时发生
<b>Load</b>	当控件加载到页面对象时发生
<b>PreRender</b>	当控件准备做输出时发生
<b>Unload</b>	当控件从内存中卸载时发生

在 **ASP.NET** 网页中，与服务器控件关联的事件在客户端（浏览器）引发，在 **Web** 服务器上处理。服务器控件仅提供有限的一组事件，通常仅限于 **Click** 类型事件。

所有事件都传递两个参数：第一个参数表示引发事件的对象，以及包含任何事件特定信息的事件对象；第二个参数通常是 **EventArgs**类型，但对于某些控件而言是特定于该控件的类型。

在服务器控件中，某些事件（通常是**Click**事件）会导致页被立即回发到服务器。而另一些事件（通常是**Change**事件）不会立即导致页被发送，它们在下次发生发送操作时引发。如果希望改变的操作立即回发到服务器，让**Change**事件导致页发送，则需要设置**Web**服务器控件的**AutoPostBack**属性。当该属性为**true**时，控件的更改事件会导致页面立即发送，而不等待**Click**事件。例如，默认情况下，**CheckBox**控件的 **CheckedChanged**事件不会导致该页被提交。但是，如果将此控件的 **AutoPostBack**属性设置为**true**，那么一旦用户单击该复选框，该页便会立即被发送到服务器进行处理。

5.3.5 标准控件详解

标准控件是 **Web** 服务器控件中最常用的空间。

1. 输入与显示控件

输入控件用来接受用户在浏览器端输入的信息，包括文本框、密码框、多行文本框，用 **TextBox** 控件可以实现它们。显示控件用来显示信息，包括标签（**Label** 控件）、文本（**Literal** 控件）。

(1) TextBox 控件

**TextBox** 控件（文本框控件）是用得最多的控件之一，该控件可以用来显示数据或者输入数据。

语法示例如下：

```
<asp:Textbox id="TextBox1" Text = "输入的数据" Column="25" MaxLengh="35" runat="server" />
```

TextBox 控件有一个重要的属性：TextMode。该属性包括 3 个选项。

- ① SingleLine：单行编辑框。
- ② MultiLine：带滚动条的多行文本框。
- ③ PassWord：密码输入框，所有输入字符都用特殊字符（如“\*”）来显示。

许多浏览器都支持自动完成功能，该功能可帮助用户根据以前输入的值向文本框中填充信息。自动完成的精确行为取决于浏览器。通常，浏览器根据文本框的 name 属性存储值。任何同名的文本框（即使是在不同页上）都将为用户提供相同的值。TextBox 控件支持 AutoCompleteType 属性，该属性提供了用于控制浏览器如何使用自动完成的选项。

除此以外，TextBox 控件还有一个常用的 TextChanged 事件，当文字改变时引发此事件，可对编写事件处理代码做出响应。

下面的代码演示了如何响应 TextBox 控件中的更改，在一个标签中显示文本框控件的内容。

```
protected void TextBox1_TextChanged(object sender, EventArgs e)
{
    Label1.Text = Server.HtmlEncode(TextBox1.Text);
}
```

默认情况下，TextChanged 事件并不马上导致向服务器回发 Web 页。而是当下次发送窗体时在服务器代码中引发此事件。若要使 TextChanged 事件引发即时发送，则需将 TextBox 控件的 AutoPostBack（自动回传）属性设置为 true。

## （2）Label 控件

Label 控件用于在 Web 页面上显示文本。其控件定义的语法示例如下：

```
<asp:Label id="Label1" Text = "密码提示回答：" runat="server" />
```

Label 控件的最常用属性为 Text，该属性表示在 Label 中显示的文本。该文本是可编程的。

## （3）Literal 控件

Literal 控件的工作方式类似于 Label 控件，用于在浏览器上显示在整个过程中不发生变化的文本。其控件定义的语法示例如下：

```
<asp:Literal id="Literal1" Text = "新用户注册" runat="server" />
```

Literal 控件支持 Mode 属性，该属性用于指定控件对所添加的标记的处理方式。可以将 Mode 属性设置为以下值。

- ① Transform：添加到控件中的任何标记都将进行转换，以适应请求浏览器的协议。
- ② PassThrough：添加到控件中的任何标记都将按原样呈现在浏览器中。
- ③ Encode：添加到控件中的任何标记都将使用 HtmlEncode 方法进行编码，该方法将把 HTML 编码转换为其文本表示形式。当希望浏览器显示而不解释标记时，编码将很有用。

【例 5-2】在网站 Chapter5 中添加一个网页 example5-2，用于体现 TextBox、Label、Literal 控件的使用。

（1）右击“解决方案资源管理器”内的项目名称，选择“添加新项”选项，再选择新建“Web 窗体”，并输入文件名“example5-2”，单击【确定】按钮。

（2）切换到“example5-2”的设计视图，从工具箱中的标准控件中拖拽 1 个 Label、1 个 TextBox 和 1 个 Literal 控件到页面上，

（3）打开 Label 控件所对应的窗口，Text 属性值设置为“为什么选择 c#：”。以同样的方

法设置 TextBox 的 TextMode 为 “MultiLine”，AutoPostBack 为 “True”，Literal 的 Mode 设置为 “Encode”。

(4) 打开 example5-2.aspx.cs 代码页，在 Page\_Load 的方法体内添加如下代码：

```
if (IsPostBack)                                //如果不是首次加载则执行下面代码
{
    string a = TextBox1.Text;                    //接受 TextBox1 所输入的内容并赋给 string 类型 a
    Literal1.Text = "<h1>" + a + "</h1>"; //在 a 中添加标题 HTML 标签
}
```

(5) 按【Ctrl+F5】组合键运行网页，在 TextBox 文本框中输入“c#是微软为 ASP.NET 贴身打造的语言”，单击页面，结果如图 5.7 所示。



图 5.7 example5-2 页面运行的结果

## 2. 按钮控件和HyperLink控件

按钮是提交窗体的常用元素，HyperLink 控件在 Web 页上创建超级链接。

### (1) 按钮控件

标准控件中包括三种类型的按钮：标准命令按钮（Button 控件）、超级链接样式按钮（LinkButton 控件）和图形化按钮（ImageButton 控件）。这三种按钮提供类似的功能，但具有不同的外观。

定义上述三种按钮的语法格式如下：

```
<asp:Button id="Button1" runat="server" Text="按钮"></asp:Button>           //Button 控件
<asp:LinkButton id="LinkButton1" runat="server">链接按钮</asp:LinkButton> //LinkButton 控件
<asp:ImageButton id="ImageButton1" runat="server" ImageUrl="..."></asp:ImageButton> //ImageButton 控件
```

三种按钮的共同属性如下。

① PostBackUrl 属性：利用这个属性可以将按钮变成“返回”按钮，即先将该属性设成某个网页的 URL，以后单击该按钮时就会直接转向该网页。

② OnClientClick 属性：定义当单击按钮时执行的客户端脚本，通常是一个脚本函数的函数名。

③ CommandName 属性：当在网页上具有多个按钮控件时，可使用 CommandName 属性来指定或确定与每一个按钮控件关联的命令名。可以用标识要执行的命令的任何字符串来设置 CommandName 属性。然后，以编程方式确定按钮控件的命令名并执行相应的操作。

三种按钮的共同事件如下。

① Click 事件：按钮的 OnClick 属性对应的值就是此事件添加的处理函数的函数名。处理函数将在服务器端执行，如果为某个按钮控件同时指定了 OnClientClick 属性和 OnClick 属性，那么将优先响应客户端的处理。

② Command 事件：当单击按钮控件时会引发 Command 事件。通常当命令名（如 Sort）与按钮控件关联时，才会使用该事件。这使得可以在一个网页上创建多个按钮控件，并以编程方式确定单击了哪个按钮控件。

当用户单击按钮控件时，该页回发到服务器。默认情况下，该页回发到其本身，重新生成相同的页面并处理该页上控件的事件处理程序。

(2) HyperLink 控件

HyperLink 可以在应用程序中的页之间移动跳转到其他页面，相当于 HTML 中的<a href>元素。

HyperLink 控件定义的语法格式如下：

```
<asp:HyperLink ID="HyperLink1" runat="server">网站服务条款</asp:HyperLink>
```

与大多数服务器控件不同的是，当用户单击 HyperLink 控件时并不会在服务器代码中引发事件。使用 HyperLink 控件的主要优点是可以通过代码动态设置链接目标。

【例 5-3】 在网站 Chapter5 中添加一个网页 example5-3，用于体现 Button、LinkButton、ImageButton 和 HyperLink 控件的使用。

① 按【例 5-2】所示的方法添加一个命名为“example5-3”的网页。打开资源管理器→右击项目名→选择“新建文件夹”选项并命名为“Image”，右击 Image 文件夹→选择“添加现有项”选项，如图 5.8 所示，添加 2 张文件名为“rm”、“百度”的图片。

② 切换到“example5-3”的设计视图，从工具箱中的标准控件中拖拽 1 个 Button、1 个 LinkButton、1 个 ImageButton 和 1 个 HyperLink 控件到页面上，

③ 打开 Button 控件所对应的属性窗口，Text 属性值设置为“显示”、CommandName 设置为“单击此按钮显示页面中其他按钮”。以同样的方法设置 LinkButton 的 Text 为“example5-2”、Visible 为“False”、PostBackUrl 为“~/example5-2.aspx”、LinkButton 的 Visible 为“False”、OnClientClick 为“showmusic()”、ImageUrl 为“~/Image/rm.gif”、HyperLink 的 Visible 为“False”、ImageUrl 为“~/Image/百度.JPG”。

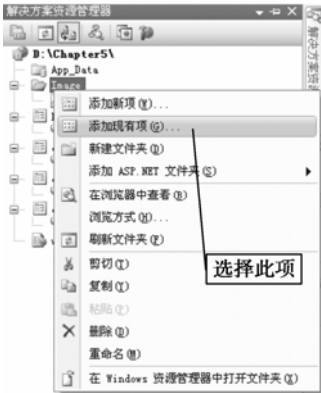


图 5.8 添加图片

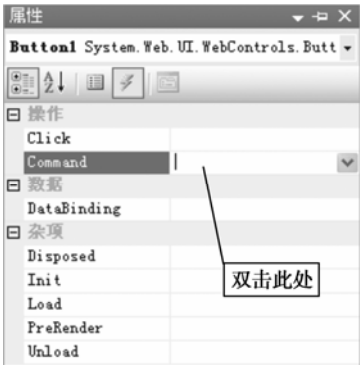



图 5.9 添加事件

④ 切换到源视图，添加脚本（黑体代码）。

```
<head runat="server">
    <title>无标题页</title>
    <script type="text/javascript">
        function showmusic()
        { alert("歌曲名：我要飞得更高"); }           //弹出对话框
    </script>
</head>
```

⑤ 打开 Button 控件的属性窗口→单击事件按钮→双击 Command 后面空白处，如图 5.9 所示。系统自动切换到 example5-3.aspx.cs 代码页并添加了方法 Button1\_Command。添加下面代码（黑体代码）：

```
protected void Page_Load(object sender, EventArgs e)
{
    HyperLink1.NavigateUrl = "http://www.baidu.com";    }
    protected void Button1_Command(object sender, CommandEventArgs e)
    {
        Response.Write("<script>alert('"+e.CommandName+"')</script>");           //显示 CommandName 属性
        LinkButton1.Visible = true;           //显示 LinkButton1
        ImageButton1.Visible = true;           //显示 ImageButton1
    }
}
```

⑥ 按【Ctrl+F5】组合键运行网页，单击【显示】按钮后弹出对话框，如图 5.10 所示，单击【确定】后页面上显示了 LinkButton、ImageButton 和 HyperLink 控件。单击 LinkButton 控件跳转到 example5-2 页面，单击 ImageButton 控件弹出对话框（如图 5.10 所示），单击 HyperLink 控件跳转到百度主页。

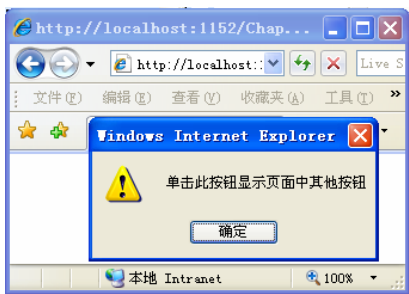


图 5.10 单击【显示】按钮后的结果

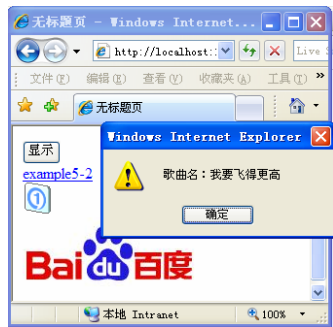


图 5.11 单击 ImageButton 控件显示的结果

### 3. 选择和列表控件

标准控件中可以给用户提供简单选择的控件有单选按钮（RadioButton）和复选框（CheckBox）。

标准控件中以列表方式呈现选项的有单选按钮列表（RadioButtonList）、复选框列表（CheckBoxList）、列表框（ListBox）、项列表（BulletedList）和下拉框（DropDownList）。

### (1) RadioButton 控件

**RadioButton** 控件表现为 Web 页面上的单选按钮。它允许用户选择 **true** 状态或 **false** 状态，但是只能选择其一。窗体上的一个单选按钮没有什么意义，在使用时通常由两个以上的 **RadioButton** 控件组成一组，以提供互相排斥的选项。页面上的一组 **RadioButton** 控件可以定义如下：

```
<asp:RadioButton ID="RadioButtonMale" runat="server" GroupName="Group1"
    Text="男" AutoPostBack="True"/>
<asp:RadioButton ID="RadioButtonFemale" runat="server" GroupName="Group1"
    Text="女" AutoPostBack="True"/>
```

### (2) RadioButtonList 控件

**RadioButtonList** 控件在 Web 页面上显示为一个单选按钮列表，用户在这一组列表项中只能选择一项。虽然多个 **RadioButton** 控件可以也组成单选按钮组以实现互斥选择，但当有多个选项供用户进行选择时，使用 **RadioButtonList** 控件更加方便。

**RadioButtonList** 控件定义示例如下：

```
<asp:RadioButtonList id="RadioButtonList1" runat="server" AutoPostBack="True">
    <asp:ListItem Value="0">男</asp:ListItem>
    <asp:ListItem Value="1">女</asp:ListItem>
</asp:RadioButtonList>
```

**RadioButtonList** 控件的 **Items** 集合的成员与列表中的每一项对应，要确定选中了哪些项，应测试每项的 **Selected** 属性。

### (3) CheckBox 控件

**CheckBox** 控件在 Web 窗体页上创建复选框。与 **RadioButton** 控件相似，**CheckBox** 控件也为用户提供了一种在二选一（如真/假、是/否或开/关）选项之间切换的方法。当用户选中该控件时，表示输入的是 **True**；当没有选中该控件时，表示输入的是 **False**。**CheckBox** 控件在使用时通常也与其他 **CheckBox** 控件组成一组，在 **CheckBox** 控件组中，用户能选择多个，也就是多选。

**CheckBox** 控件定义示例如下：

```
<asp:CheckBox ID="CheckBox1" runat="server" Text="我已阅读并同意遵守网站服务条款" />
```

### (4) CheckBoxList 控件

**CheckBoxList** 控件提供给用户一个复选框列表，它相当于一个 **CheckBox** 控件组。**CheckBox** 控件允许操作一个条目，而 **CheckBoxList** 控件允许操作的是一组条目。**CheckBox** 控件可提供对布局的更多控制，而 **CheckBoxList** 控件提供方便的数据绑定功能。

**CheckBoxList** 控件定义示例如下：

```
<asp:CheckBoxList id="CheckBoxList1" runat="server">
    <asp:ListItem Value="琴">琴</asp:ListItem>
    <asp:ListItem Value="棋">棋</asp:ListItem>
    <asp:ListItem Value="书">书</asp:ListItem>
    <asp:ListItem Value="画">画</asp:ListItem>
</asp:CheckBoxList>
```

### (5) DropDownList 控件

DropDownList 控件在 Web 页面上呈现为下拉列表框，它允许用户从预定义的多个选项中选择一项。在选择前，用户只能看到第一个选项，其余的选项都“隐藏”起来。

DropDownList 控件定义示例如下：

```
<asp:DropDownList id="DropDownList1" runat="server">
    <asp:ListItem Value="0">博士</asp:ListItem>
    <asp:ListItem Value="1">硕士</asp:ListItem>
    <asp:ListItem Value="2">本科</asp:ListItem>
    <asp:ListItem Value="3">专科</asp:ListItem>
</asp:DropDownList>
```

### (6) ListBox 控件

ListBox 控件表示在一个滚动窗口中垂直显示的一系列项目列表。ListBox 允许选择单项或多项，并通过常见的 Items 集合提供内容。与 DropDownList 类似，列表框 ListBox 可以实现从预定义的多个选项中进行选择的功能。区别在于，ListBox 在用户选择操作前，可以看到所有的选项，并可以实现多项选择。

ListBox 控件定义示例如下：

```
<asp:ListBox ID="ListBox1" runat="server">
    <asp:ListItem Value="0">已工作</asp:ListItem>
    <asp:ListItem Value="1">大学生</asp:ListItem>
    <asp:ListItem Value="2">中学生/中专技校</asp:ListItem>
    <asp:ListItem Value="3">以上都不是</asp:ListItem>
</asp:ListBox>
```

Rows 属性用来获取或设置 ListBox 控件中所显示的行数。SelectionMode 属性用来控制是否支持多行选择，如果将 ListBox 控件设置为多选，则用户可以在安装【Ctrl】或【Shift】键的同时，单击以选择多个选项。

### (7) BulletedList 控件

BulletedList 控件创建一个无序或有序（编号的）项列表，可以指定项、项目符号或编号的外观，静态定义列表项或通过将控件绑定到数据来定义列表项，也可以在用户单击项时做出响应。

BulletedList 控件的定义示例如下：

```
<asp:BulletedList ID="BulletedList1" BulletStyle="Circle" runat="server">
    <asp:ListItem>第一项</asp:ListItem>
    <asp:ListItem>第二项</asp:ListItem>
    <asp:ListItem Text="第三项"></asp:ListItem>
    <asp:ListItem Text="第四项" Value="4"></asp:ListItem>
</asp:BulletedList>
```

DisplayMode 属性确定如何显示每个项目符号的内容：纯文本（默认）、链接按钮或超链接。



【例 5-4】 在网站 Chapter5 中添加一个网页 example5-4，用于体现选择和列表控件的使用。

- ① 按【例 5-2】所示的方法添加一个命名为“example5-4”的网页。
- ② 切换到“example5-4”的设计视图，插入 8 行 2 列的表到页面中，从工具箱中的标准控件中拖拽 1 个 TextBox、2 个 RadioButton、1 个 RadioButtonList、1 个 CheckBox、1 个 DropDownList、1 个 CheckBoxList、1 个 ListBox 和 1 个 BulletedList 控件到页面上，如图 5.12 所示。
- ③ 控件的属性设置如表 5.4 所示。

表 5.4 控件的属性设置

控 件	属性: 属性值
TextBox	ID: TextBox1
RadioButton	ID : RadioButton1, Text: 男
	ID : RadioButton2 , Text: 女
RadioButtonList	ID: RadioButtonList1, Items: 博士、硕士、本科、专科
CheckBox	ID: CheckBox1, Text: 已婚
DropDownList	ID: DropDownList1, Items: 党员、团员
CheckBoxList	ID: CheckBoxList1, Items: 登山、钓鱼、篮球、音乐
ListBox	ID: ListBox1, Visible: False, SelectionMode: Multiple, Items: 看小说、逛街、跳舞、音乐
BulletedList	ID: BulletedList1, DisplayMode: HyperLink

BulletedList 控件的 Items 设置如图 5.13 所示。Text 分别为“百度”、“谷歌”、“example5-3”，Value 分别为“http://www.baidu.com”、“http://www.google.cn”、“example5-3.aspx”。在其他控件的 Items 设置中，Value 值和 Text 值设置为同一个值。

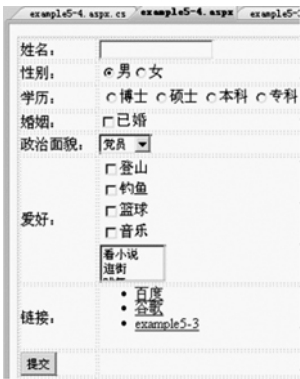


图 5.12 example5-4 页面设计



图 5.13 BulletedList 控件的 Items 设置

- ④ 分别双击 RadioButton1、RadioButton2 和 Button1 控件，添加事件代码（黑体部分）：

```
protected void RadioButton1_CheckedChanged(object sender, EventArgs e)
{
    CheckBoxList1.Visible = true;           //CheckBoxList1 可见
    ListBox1.Visible = false;               //ListBox1 不可见
}
```

```

}
protected void RadioButton2_CheckedChanged(object sender, EventArgs e)
{
    CheckBoxList1.Visible = false;           //CheckBoxList1 不可见
    ListBox1.Visible = true;                 //ListBox1 可见
}
protected void Button1_Click(object sender, EventArgs e)
{
    string name = TextBox1.Text;
    string sex;
    if (RadioButton1.Checked == true)
    { sex = "男"; }
    else
    { sex = "女"; }
    string SchRecord = RadioButtonList1.SelectedValue;
    string Marriage;
    if (CheckBox1.Checked == true)
    { Marriage = "已婚"; }
    else
    { Marriage = "未婚"; }
    string hobby=string.Empty;
    for (int i = 0; i < CheckBoxList1.Items.Count; i++)    //遍历 CheckBoxList 控件内所有项
    {
        if (CheckBoxList1.Items[i].Selected)
            hobby += CheckBoxList1.Items[i].Text + "    ";
    }
    for (int i = 0; i < ListBox1.Items.Count;i++ )
    {
        if (ListBox1.Items[i].Selected)
            hobby += ListBox1.Items[i].Text + "    ";
    }
    string s="姓名:"+name+"<br>性别:"+sex+"<br>学历:"+SchRecord+"<br>婚姻:"+Marriage+
        "<br>爱好: "+hobby;
    Response.Write(s);
}

```

⑤ 按【Ctrl+F5】组合键运行网页，填写资料，单击【提交】按钮，结果如图 5.14 所示。

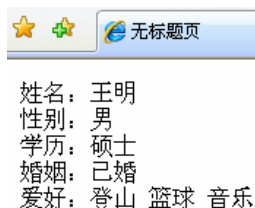


图 5.14 网页运行的部分截图

4. AdRotator控件

AdRotator 控件常用于在页面上显示广告。可以从一条和多条广告记录的数据源读取广告信息，它从列表中随机显示一个图片，这个列表可以是存储在单独的 XML 文件或数据绑定的数据源中的。

AdRotator 控件定义示例如下：

```
<asp:AdRotator id="AdRotator1" runat="server" Width="100px" Height="60px"
    AdvertisementFile=".\ad\adXml.xml">
</asp:AdRotator>
```

广告文件是一个 XML 文件，它包含了 AdRotator 控件显示的广告的有关信息。该文件的位置和文件名由控件的 AdvertisementFile 属性指定。

由于是一个 XML 文件，所以广告文件是一个使用已定义好的、使用标签描述数据的结构化文本文件。表 5.5 列出了标准标签，它们都包含在尖括号（< >）中，并需要一个匹配的关闭标签。

表 5.5 在广告文件中使用的 XML 标签

标 签	说 明
Advertisements	包含整个广告文件
Ad	描述每一个单独的广告
ImageUrl	要显示的图像的 URL，必需
NavigateUrl	单击该控件时定位到的 URL
AlternateText	图像不可用时要显示的文本。在某些浏览器中，该文本显示为工具提示
Keyword	广告类别。该关键字可用于通过设置 KeywordFilter 属性过滤要显示的广告
Impressions	一个值，指示相对于 XML 文件中的其他广告，该广告显示的频率

【例 5-5】 在网站 Chapter5 中添加一个网页 example5-5，用于体现 AdRotator 控件的使用。

- ① 添加一个命名为“example5-5”的网页。添加 1 张命名为“baidu.JPG”和 1 张命名为“google.JPG”的图片到 Image 文件夹。
- ② 通过解决方案管理器添加一个 Ad.xml 文件，把文件的内容改成如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
  <Ad>
    <ImageUrl>~/image/baidu.JPG</ImageUrl>
    <NavigateUrl>http://www.baidu.com</NavigateUrl>
    <AlternateText>百度</AlternateText>
    <Impressions>50</Impressions> //广告显示的频率
  </Ad>
  <Ad>
    <ImageUrl>~/image/google.JPG</ImageUrl>
    <NavigateUrl>http://www.google.cn</NavigateUrl>
    <AlternateText>谷歌</AlternateText>
```

<Impressions>50</Impressions> //广告显示的频率

</Ad>

</Advertisements>

- ③ 切换到“example5-5”的设计视图，从工具箱中拖拽 1 个 AdRotator 控件到页面。AdvertisementFile 属性设置为“~/Ad.xml”。
- ④ 按【Ctrl+F5】组合键运行网页，按【Ctrl+R】组合键刷新网页，所显示的图片会自动改变，单击图片链接到百度或者谷歌页面，如图 5.15 所示。



图 5.15 AdRotator 控件运行效果

5. Calendar控件

Calendar 控件在 ASP.NET 网页中显示一个单月份日历。用户可使用该日历查看和选择日期。该控件基于 DateTime 对象，因此支持该对象所允许的全部日期范围，可有效地显示从公元 0 年至 9999 年之间的任意日期。

Calendar 控件的语法示例如下：

<asp:Calendar ID="MyCalendar" runat="server"></asp:Calendar>

Calendar 控件的常用属性如表 5.6 所示。

表 5.6 Calendar 控件的常用属性

属 性	说 明
SelectedDate	被选中的日期
SelectionMode	用户被允许选择日期的方式。可使用下列值之一：None（无）、Day（日）、DayWeek（日、星期）、DayWeekMonth（日、星期、月）
SelectionDayStyle	被选中日的格式
TadayDayStyle	当天日期的样式

【例 5-6】 在网站 Chapter5 中添加一个网页，用于体现 Calendar 控件的使用。

- ① 添加一个命名为“example5-6”的网页。在设计视图状态下从工具箱中拖曳一个 Calendar 控件到页面。
- ② 根据个人喜好调整 Calendar 控件的风格，双击 Calendar 控件，添加 SelectionChanged 事件处理代码（黑体部分）：

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    Response.Write("你选择的日期为: " + this.Calendar1.SelectedDate.ToShortDateString());
}
```

③ 按【Ctrl+F5】组合键运行网页，单击其中一个日期，结果如图 5.16 所示。

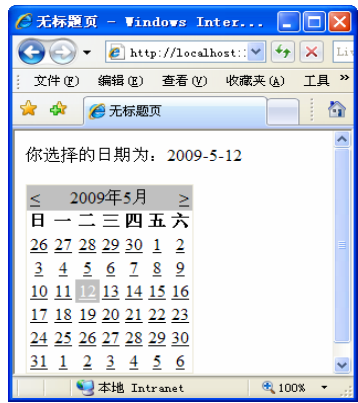


图 5.16 example5-6 网页运行的结果

## 6. Image、ImageMap和Panel控件

Image 控件用于简单地显示图像。ImageMap 控件用于创建客户端的、可单击的图像映射。

### (1) Image 控件

图像服务器控件 Image 可以在 Web 窗体页上显示图像，并用服务器端的代码管理这些图像。

Image 控件语法示例如下：

```
<asp:Image ID="Image1" runat="server" />
```

与大多数其他 Web 服务器控件不同，Image 控件不支持鼠标单击（Click）事件。需要使用鼠标单击事件时，可以使用 ImageButton 控件来代替 Image 控件。AlternateText 属性是指当有的浏览器不支持加载图像时，替代图像的文本。

### (2) ImageMap 控件

ImageMap 控件可以用来显示图像，也可以实现图像的超链接。该控件的最大特点是，可以将 ImageMap 中的图像按照（X,Y）坐标划分成不同形状的区域，分别链接到不同的网页。

ImageMap 控件语法示例如下：

```
<asp: ImageMap id="ImageMap1" runat="server" ImageUrl="~/image1.jpg"></asp: ImageMap>
```

ImageUrl 属性用来链接图像源文件；HotSpot 属性用来划分链接区域；HotSpotMode 属性是热点模式，其枚举值如表 5.7 所示。

表 5.7 HotSpotMode 枚举值

枚举值	说明
NotSet	未设置。虽然名为未设置，但默认情况下会执行定向操作，定向到指定的 URL 地址。如果未指定 URL 地址，则将定向到 Web 应用程序根目录
Navigate	定向操作。定向到指定的 URL 地址。如果未指定 URL 地址，则默认将定向到 Web 应用程序根目录
PostBack	回发操作。单击热点区域后，将执行 Click 事件
Inactive	无任何操作，即此时 ImageMap 如同一张没有热点区域的普通图片

(3) Panel 控件

Panel 控件在 Web 窗体页内提供了一种容器控件，可以将它用作静态文本和其他控件的父级。

Panel 控件语法示例如下：

```
<asp:Panel ID="Panel1" runat="server" Height="25px" Visible="False" Width="342px"></asp:Panel>
```

【例 5-7】 在网站 Chapter5 中添加一个网页，用于体现 Image、ImageMap 和 Panel 控件的使用。

① 添加一个命名为“example5-7”的网页。在设计视图状态下从工具箱中拖拽 1 个 ImageMap 控件和一个 Panel 控件到页面，在 Panel 控件中添加 1 个 Image 控件和 1 个 Label 控件。

② 添加 1 张命名为“Smil.JPG”和 1 张命名为“mq.JPG”的图片到 Image 文件夹中。Image 控件的 ImageUrl 为“~/Image/Smil.JPG”，ImageMap 控件的 ImageUrl 为“~/Image/mq.JPG”，HotSpotMode 设置为“PostBack”。ImageMap 控件的 HotSpot 设置如图 5.17 所示，单击【添加】按钮中的 ▾→选择“RectangleHotSpot”，其属性设置如表 5.8 所示。

表 5.8 RectangleHotSpot 属性设置

成 员	属 性
0	Bottom: 220; Right: 80; AlternateText: 米奇; PostBackValue: 米奇
1	Bottom: 220; Left: 81; Right: 161; AlternateText: 米妮; PostBackValue: 米妮

③ 双击 ImageMap 控件，添加 Click 事件代码，代码如下（黑体部分）：

```
protected void ImageMap1_Click(object sender, ImageMapEventArgs e)
{
    Panel1.Visible = true;                                //显示 Panel 控件
    //给容器中的 Label 控件赋值
    ((Label)Panel1.FindControl("Label1")).Text = "你喜欢" + e.PostBackValue;
}
```

④ 按【Ctrl+F5】组合键运行网页，单击“米奇”，结果如图 5.18 所示。



图 5.17 HotSpot 设置



图 5.18 单击“米奇”后的页面

## 7. MultiView和View控件

View 控件是视图控件，MultiView 控件是多视图控件，两者都属于容器控件。View 控件是一个 Web 控件的容器，而 MultiView 控件又是 View 控件的容器，因此两者多半一起搭配运作。在 MultiView 控件中可以拖曳多个 View 控件，而 View 控件内包含了任何需要显示在页面中的内容。虽然 MultiView 中可包含多个 View 控件，但页面一次只能显示一个视图。MultiView 通过 ActiveViewIndex 属性值来决定哪个 View 要被显示的，程序也是利用 ActiveViewIndex 属性设置来切换不同的 View 的。MultiView 与 View 控件关系如图 5.19 所示。

View 和 MultiView 控件语法示例如下：

```
<asp:MultiView ID="MultiView1" runat="server">
  <asp:View ID="View1" runat="server"></asp:View>
  <asp:View ID="View2" runat="server"></asp:View>
</asp:MultiView>
```

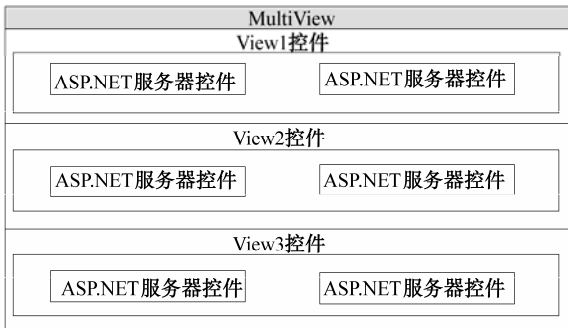


图 5.19 MultiView 与 View 控件关系图

MultiView 控件包含 ActiveViewIndex 属性，该属性可获取或设置以 0 开始的当前活动视图的索引。如果没有视图是活动的，那么 ActiveViewIndex 为默认值-1。视图索引值是指在 MultiView 控件中的设置先后次序，以 0 为起始值。

**【例 5-8】** 在网站 Chapter5 中添加一个网页，用于体现 MultiView 和 View 控件的使用。

① 添加一个命名为“example5-8”的网页。在设计视图状态下从工具箱中拖曳 1 个 RadioButtonList 控件和一个 MultiView 控件到页面，拖曳 2 个 View 控件到 MultiView 控件中。

② 在页面上输入提示语，控件的属性设置如表 5.9 所示，设计后的页面如图 5.20 所示。

表 5.9 example5-8 页面控件的属性设置

控 件	属 性 设 置
RadioButtonList	AutoPostBack: True; Items: 男 (Selected: False、Text: 男、Value: 男)、女(Selected: False、Text: 女、Value: 女)
ListBox	Items: 唱歌、跳舞、逛街、看电影; SelectionMode: Multiple
CheckBoxList	Items: 篮球、足球、垂钓、音乐

③ 双击 RadioButtonList 控件，添加 SelectedIndexChanged 事件代码（黑体部分）：

```
protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (RadioButtonList1.SelectedValue == "女")
    {   MultiView1.ActiveViewIndex = 0;   }           //获取视图的索引
    else
    {   MultiView1.ActiveViewIndex = 1;   }
}
```

④ 按【Ctrl+F5】组合键运行网页，根据选择性别“女”或“男”提供不同的选项，如图 5.21 所示。

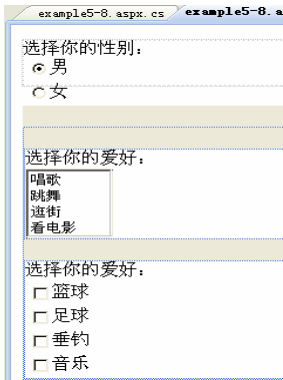


图 5.20 页面设计部分截图

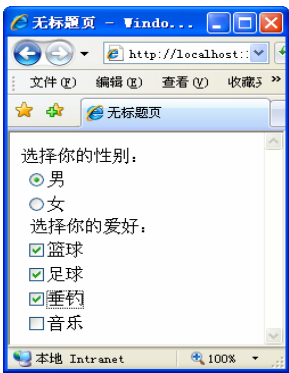


图 5.21 选择“男”后的网页运行结果

8. FileUpload控件

FileUpload 控件允许用户把文件上传到 Web 服务器上。它包含一个“浏览”按钮和一个用于输入文件名的文本框。只要用户在文本框中输入了完全限定的文件名，则无论直接输入或通过“浏览”按钮选择，都可以通过调用 FileUpload 的 SaveAs 方法保存到磁盘上。

用户选择要上传的文件后，FileUpload 控件不会自动将该文件保存到服务器，而必须显式提供一个控件或机制，使用户能提交指定的文件。例如，可以提供 一个按钮，用户单击它即可上传文件。

可以通过 FileUpload 的一些重要属性访问上传的文件。

(1) FileName 属性：用来获取客户端上使用 FileUpload 控件上传的文件名称。



(2) FileBytes 属性：该属性从使用 FileUpload 控件指定的文件返回一个字节数组，包含了指定文件的内容。

(3) FileContent 属性：该属性获取 Stream 对象，该对象指向使用 FileUpload 控件上传的文件。可以使用 FileContent 属性来访问文件的内容。例如，可以使用该属性返回的 Stream 对象以字节方式读取文件内容并将它们存储在一个字节数组中。

9. Wizard控件

Wizards 控件为用户提供了呈现一连串步骤的基础架构，这样可以访问所有步骤中包含的数据，并方便地进行前后导航。Wizard 控件主要的功能是提供导航和用户接口（UI），以收集多个步骤中的相关信息。

Wizard 控件可区分成四大区域。

(1) 向导步骤（WizardStep）区域：Wizard 控件使用多个步骤来描绘用户输入的不同部分。每个步骤的内容添加在标记<asp:WizardStep>中，实际应用时，每次只能显示一个<asp:WizardStep>定义的内容。

(2) 标题（Header）区域：用于在步骤顶部提供一致的信息，此项是可选元素。

(3) 侧栏（SideBar）区域：此项也是可选元素，通常显示在向导左边，包含所有步骤的列表，并提供在各个步骤间的跳转。


(4) 导航按钮（Navigation）区域：Wizard 内置导航功能，它会根据步骤类型（Step Type）设置值的不同，而呈现不同的导航按钮。

每个 WizardStep 步骤都会有一个 StepType 属性，其最主要的作用是决定每个步骤中的导航 Button 按钮如何被显示。StepType 的类型说明如表 5.10 所示。

表 5.10 StepType 的类型说明

StepType 类型	说 明
Start（开始步骤）	这是第一个开始步骤，只会呈现【下一步】按钮
Step（阶段步骤）	在 Start 及 Finish 之间的步骤全部归类为 Step，Step 会同时呈现【上一步】及【下一步】按钮
Finish（完成步骤）	这是最后的数据收集步骤，会呈现【完成】及【上一步】按钮，但若前一个步骤的 AllowReturn 设置为 False，则不显示【上一步】按钮
Complete（结束步骤）	这是 Wizard 的最后一个步骤画面，完全不会呈现任何按钮，甚至连 SideBar 区域也会消失；若就英文字面则很难区分 Complete 和 Finish 的差别，但就实质而言，Complete 较贴近最后的结束
Auto（自动）	系统会依该步骤的顺序决定其为何种 StepType 类型

【例 5-9】 在网站 Chapter5 中添加一个网页，用于体现 FileUpload 和 Wizard 控件的使用。

① 添加一个命名为“example5-9”的网页。在设计视图状态下从工具箱中拖拽 1 个 Wizard 控件到页面上，将 Wizard 拉到一定的大小，在 Wizard 控件的属性窗口中选择 WizardSteps，单击后面的“”图标，添加 1 个成员，Title 为“Step3”，StepType 为“Complete”。

② 单击“Step1”，添加 1 个 TextBox 放在 Wizard 中，前面输入“姓名”字样，如图 5.22 所示。用同样的方法单击“Step2”，添加 1 个 FileUpload 控件、1 个 Button 控件、1 个 Label 控件到 Wizard 上，Button 的 Text 属性设置为“上传”。单击“Step3”，添加 1 个 Label 控件和 1 个 Image 控件到 Wizard 上。

③ 打开 Wizard 控件的属性窗口，双击 FinishButton 事件后的空白处，系统自动添加方法 “Wizard1\_FinishButtonClick”。在 “Step2” 中双击 Button 控件，系统自动添加方法 “Button\_Click”。在代码页中添加如下代码：

```
static String imageName; //声明静态的 string 类型变量 imageName 存放图片名
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
        Wizard1.ActiveStepIndex = 0; //首次加载网页时呈现第一步
}
protected void Button1_Click(object sender, EventArgs e)
{
    bool fileOK = false;
    if (FileUpload1.HasFile)
    {
        String fileExtension = System.IO.Path.GetExtension(FileUpload1.FileName).ToLower();
        String[] allowedExtensions = { ".gif", ".png", ".jpeg", ".jpg" };
        for (int i = 0; i < allowedExtensions.Length; i++)
            //判断文件后缀名是否是指定图片类型
            if (fileExtension == allowedExtensions[i])
                fileOK = true;
    }
    if (fileOK)
    {
        try
        {
            String path = Server.MapPath("~/Image/"); //获取 Image 文件夹的物理地址
            FileUpload1.PostedFile.SaveAs(path + FileUpload1.FileName); //上传图片
            Label1.Text = "上传成功! ";
            imageName = FileUpload1.FileName;
        }
        catch (Exception ex)
        { Label1.Text = "不能上传此文件"; }
    }
    else
        Label1.Text = "文件类型不正确";
}
protected void Wizard1_FinishButtonClick(object sender, WizardNavigationEventArgs e)
{
    //单击【完成】按钮所执行的操作
    Label2.Text = TextBox1.Text;
    Image1.ImageUrl = "~/Image/" + imageName; //显示图片
}
```

④ 按【Ctrl+F5】组合键运行网页，根据选择性别“女”或“男”提供不同的选项，如图 5.23 所示。



图 5.22 在 Wizard 中添加控件



图 5.23 example5-9 网页运行的结果

## 5.4 验证控件

在 Internet 上收集数据时，为确保所收集的数据有价值、有意义，应避免所收集的数据违反制定的规则。验证控件是一系列控件，可以处理终端用户在应用程序的窗体元素中输入的信息。这些控件可确保放在窗体上的数据的有效性。验证控件位于 VS 2008 的“工具箱”的“验证”选项卡中。

### 5.4.1 客户端验证和服务端验证

在窗体回送给服务器之前，对输入该窗体中的数据进行的验证称为客户端验证。当请求发送到应用程序所在的服务器后，在请求、响应循环的这一刻，就可以为所提交的信息进行有效性验证，这称为服务器端验证。

验证工作最好放在客户端进行。在客户端输入完数据，向服务器提交之前应对数据进行检测，如果发现错误，应立即提示并要求改正，改正前不向服务器提交信息。这种处理方式可以将改正错误的过程放在提交以前，减少网上的无效传输。但是有两个原因使客户端验证不可依赖：

- (1) 相当一部分客户端的设备功能弱，不具备验证能力，此时验证工作只能放在服务器端进行；
- (2) 恶意的用户能够比较容易地破坏客户端的验证脚本，或者想方设法绕过客户端的校验。

因此，从安全角度出发，不论客户端是否进行了验证，服务器端的验证都是不可缺少的，除非人为地取消了服务器端验证。当用户向服务器提交数据之后，服务器都毫无例外地调用验证程序来逐个检查用户的输入。如果发现任何输入数据有错误，则整个页面将自行设置为无效状态，并发出错误信息。

### 5.4.2 验证控件分类及作用

ASP.NET 提供了 6 种验证控件。

- (1) RequiredFieldValidator 控件：检查用户是否在输入控件中输入了数据。
- (2) CompareValidator 控件：将输入控件的值与常数值或其他输入控件的值相比较，以确定这两个值是否与由比较运算符（小于、等于、大于等）指定的关系相匹配。

- (3) **RangeValidator** 控件：检查用户的输入是否在一个特定的范围内。
- (4) **RegularExpressionValidator** 控件：检查用户的输入是否与正则表达式所定义的模式匹配。
- (5) **CustomValidator** 控件：通过用户定义的验证函数判定输入的数据是否有效。
- (6) **ValidationSummary** 控件：以列表的形式显示页面上所有验证控件所搜索到的验证错误。

其中，**ValidationSummary** 控件只能与前 5 种验证控件一起使用，不能单独执行验证。另外在这些控件中，除 **RequiredFieldValidator** 控件以外，其他所有的控件都认为空字段是合法的。

### 5.4.3 验证控件详解

各个控件虽然作用不同，但使用的方法却有很多共同点。每个控件都有一个 **ControlToValidate** 属性，必须用它来指定被验证的控件名称。下面分别介绍各验证控件的使用方法。

#### 1. RequiredFieldValidator控件

**RequiredFieldValidator** 控件用于对一些必须输入的信息进行检验，如果一些必须输入的数据没有输入，则将提示错误。

使用这个控件的方法比较简单，将控件拖入窗体以后，关键是给它设置以下 4 个属性。

- (1) **ControlToValidate**：设置被验证的控件，可以在该属性的下拉列表中选择。
- (2) **ErrorMessage**：当不能通过验证时显示的错误信息。
- (3) **Display**：显示错误信息的位置，包括以下 3 种选择。
  - ① **None**：不显示错误信息。
  - ② **Static**：显示在设计时控件所放置的位置。
  - ③ **Dynamic**：将错误信息动态显示在页面上。
- (4) **EnableClientScript**：该属性为逻辑变量，默认为 **true**，表示如有可能（如浏览器版本为 Internet Explorer 4.0 以上），则先在客户端验证。若将该属性值改为 **false**，则将不在客户端进行验证。

#### 2. CompareValidator控件

**CompareValidator** 控件用来将输入到控件（如 **TextBox** 控件）的值与输入到其他控件的值或常数值进行比较。其几个重要属性的设置方法如下。

- (1) 通过设置 **ControlToValidate** 属性指定被验证的控件 ID。
- (2) 如果将输入控件与其他控件值进行比较，则将 **ControlToCompare** 属性设置为要与之相比较的控件 ID。如果将输入控件的值与某个常数值进行比较，则应将 **ValueToCompare** 属性设置为与之相比较的常数。
- (3) **Type** 属性用于设置比较数据的类型，只有在同一类型的数据之间才能够进行比较。
- (4) **Operator** 属性用来指定比较的方法，如大于、等于等。如果将 **Operator** 属性设置为 **DataTypeCheck**，则 **CompareValidator** 控件将忽略 **ControlToCompare** 和 **ValueToCompare** 属性，并且仅仅指示输入到输入控件中的值是否可以转换为 **Type** 属性指定的数据类型。

### 3. RangeValidator控件

RangeValidator 控件用于检查用户的输入是否在一个特定的范围内，可以检查数字对、字母对和日期对的限定范围，边界表示为常数。

RangeValidator 控件的主要属性如下。

- (1) ControlToValidate 属性：指定被验证的输入控件。
- (2) MinimumValue 属性和 MaximumValue 属性：分别指定有效范围的最小值和最大值。
- (3) Type 属性：用于指定要比较的值的数据类型。

### 4. RegularExpressionValidator控件

RegularExpressionValidator 控件用来验证输入的格式是否匹配某种特定的模式（正则表达式）。这类验证允许检查一些可以预知的字符序列，如身份证号码、电子邮件地址、电话号码和邮编中的字符序列等。除非浏览器不支持客户端验证，或者已明确禁止客户端验证（将 EnableClientScript 属性设置为 false），否则将同时执行服务器端验证和客户端验证。

客户端的正则表达式验证实现与服务器端的略有不同。在客户端，使用的是 JScript 正则表达式语法。而在服务器端，使用的则是 System.Text.RegularExpressions.Regex 语法。由于 JScript 正则表达式语法是 System.Text.RegularExpressions.Regex 语法的子集，所以最好使用 JScript 正则表达式语法，以便在客户端和服务器端得到同样的结果。

在使用本控件进行校验时，除按照前面几个控件设置属性以外，最主要的区别是对控件的 ValidationExpress 属性设置检查模式。方法是单击属性右边的省略号按钮，在弹出的对话框中选择“标准表达式”，又弹出的对话框如图 5.24 所示，然后选择需要检查的模式即可。

**注意：**在编写处理字符串的程序或网页时，经常会有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些规则的工具。换句话说，正则表达式就是记录文本规则的代码。正则表达式具有复杂的语法定义，感兴趣的读者可以登录 <http://unibetter.com/deerchao/zhengzhe-biaodashi-jiaocheng-se.htm>查看有关内容。

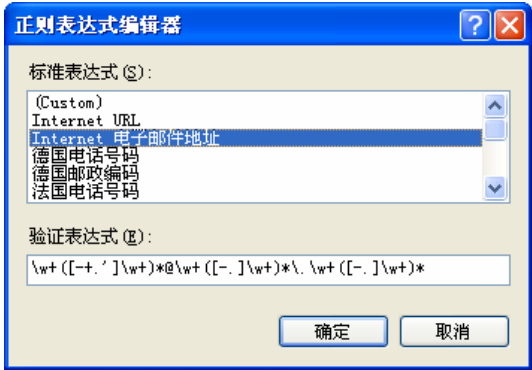


图 5.24 “正则表达式编辑器”对话框

### 5. CustomValidator控件

使用自定义控件 CustomValidator 时，可以自行定义验证算法，并同时利用控件提供的其

他功能。

为了在服务器端验证函数，先将 CustomValidator 控件拖入窗体，并将 ControlToValidate 属性指向被验证的对象，然后给该验证控件的 ServerValidate 事件提供一个验证程序，最后在 ErrorMessage 属性中填写出现错误时显示的信息。

在 ServerValidate 事件处理程序中，可以从 ServerValidateEventArgs 参数的 Value 属性中获取输入到被验证控件中的字符串。验证的结果要存储到 ServerValidateEventArgs 的属性 IsValid (true 或 false) 中。

例如，利用自定义 CustomValidator 控件验证某个输入框中输入的数据能否被 3 整除，若不能被 3 整除则发出错误信息。事件处理的代码如下：

```
private void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs args)
{
    int number=int.Parse(args.Value);           //取出输入的数据
    if((number % 3) == 0)                        //校验能否被 3 整除
        args.IsValid=true;                     //结果正确
    else
        args.IsValid=false;                     //结果错误
}
```

如果需要同时提供客户端验证程序以便让具有 DHTML 能力的浏览器先进行验证，则应该在.aspx 的 HTML 视图中用 JavaScript 语言编写验证程序，同时将验证的函数名写入控件的 ClientValidationFunction 属性中。

6. ValidationSummary控件

ValidationSummary 控件用于在一个位置上集中显示来自 Web 网页上所有验证程序的错误信息。根据 DisplayMode 属性的设置，可以采用列表、项目符号列表或单个段落的形式来显示。通过设置控件的 ShowSummary 和 ShowMessageBox 属性，可以确定显示的形式。

【例 5-10】 在网站 Chapter5 中添加一个网页，用于体现验证控件的使用。

- ① 添加一个命名为“example5-10”的网页。切换到设计视图下，选择菜单命令“表”→“插入表”，添加 1 个 6 行 3 列的表，在设计视图状态下从工具箱中拖拽 5 个 TextBox 控件到页面上。
- ② 表的第一列输入提示字样。表的第二列分别添加 5 个 TextBox 控件，ID 从上到下分别为“TextBox1”、“TextBox2”、“TextBox3”、“TextBox4”、“TextBox5”。表的第三列和表的第 6 行添加 1 个 Button 控件和一些验证控件，如图 5.25 所示。其中 RequiredFieldValidator 控件的 ID 与同行的 TextBox 控件一致，ControlToValidate 属性为同行的 TextBox 控件的 ID，ErrorMessage 分别设置为“用户名不能为空”、“请输入密码”、“请输入确认密码”、“请输入年龄”、“请输入邮箱”。

③ 页面上控件的其他属性设置如表 5.11 所示。其中，ID 为 RegularExpressionValidator1 的 ValidationExpression 的设置如图 5.24 所示。属性设置完成后的页面如图 5.26 所示。

④ 打开 CustomValidator 控件的属性窗口，单击图标，双击 ServerValidate 后的空白处，系统自动在代码页中添加了方法“CustomValidator1\_ServerValidate”。在此方法体中添加如下代码：

```
int a = 0;
```

```
foreach (char s in args.Value) //遍历输入密码，密码的个数存放在变量 a 中
{
    a++;
}
if (a < 6 || a > 15) //密码个数是否在 6 与 15 之间
{
    args.IsValid = false; //结果正确
}
else //结果错误
{
    args.IsValid = true;
}
```

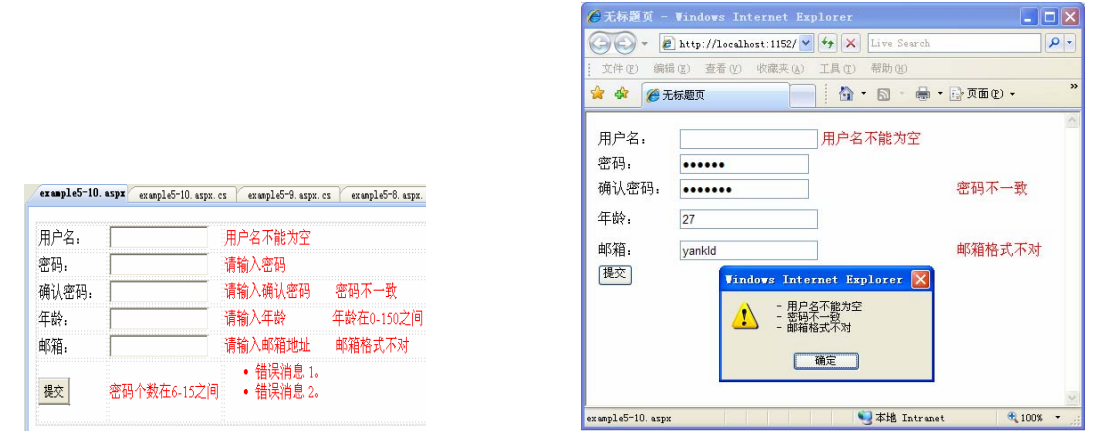


图 5.25 example5-10 页面布局与空间添加

表 5.11 其他属性设置

控件 ID	属 性 设 置
TextBox2、TextBox3	TextMode: Password
CompareValidator1	ControlToCompare: TextBox2; ControlToValidate: TextBox3; ErrorMessage: 密码不一致
RangeValidator1	ControlToValidate: TextBox4; ErrorMessage: 年龄在 0~150 之间; MinmumValue: 0; MaximumValue: 150; Type: Integer
RegularExpressionValidator1	ControlToValidate: TextBox5; ErrorMessage: 邮箱格式不对、
CustomValidator1	ControlToValidate: TextBox2; ErrorMessage: 密码个数在 6~15 之间、
ValidationSummary1	ShowMessageBox: True; ShowSummary: False

⑤ 按【Ctrl+F5】组合键运行网页，输入一些数据，单击【提交】按钮，如图 5.27 所示，无法进行提交。只有输入合法后，【提交】按钮才可以提交，并运行后台的代码。



### 5.4.4 关闭客户端验证功能

验证服务器控件会自动为客户机提供客户端验证功能，前提是请求容器可以正确处理所生成的 JavaScript。但有时需要控制客户端验证，要关闭验证控件的客户端验证功能，让它们不再把客户端验证功能发送给请求者。

关闭客户端验证功能有两种方式。

(1) 用编程方式删除验证控件的客户端验证功能。例如，在 Page\_Load 事件中关闭页面上所有验证控件的客户端脚本功能。如果要动态地确定不允许进行客户端验证，则使用此方法比较好：

```
protected void Page_Load(object sender, EventArgs e)
{
    foreach(BaseValidator Bv in Page.Validators)
    {
        Bv.EnableClientScript=false;
    }
}
```

(2) 设置验证服务器控件的 EnableClientScript 属性为 Flase，从而阻止控件发送给在客户机上执行验证的 JavaScript 函数，使验证检查在服务器上进行。该属性默认为 True。此属性的用法如下：

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" ControlToValidate=
"TextBox1"
Text="必填" EnableClientScript="false">
```

## 5.5 用户控件

虽然 ASP.NET 服务器控件提供了大量的功能，但它们并不能涵盖每一种情况，不可能完全满足程序设计人员的所有需求。在 ASP.NET 中，可以自定义控件（用户控件）以方便程序的设计。

一个 Web 用户控件与一个完整的 Web 窗体页相似，它们都包含一个用户界面页和一个代码隐藏文件。在用户控件上可以使用与标准 Web 窗体页上相同的 HTML 元素和 Web 控件。但与.aspx 文件不同的是：用户控件界面页定义在扩展名为.ascx 的文件中，不能作为独立的 Web 窗体页来运行；用户控件中不包含<HTML>、<BODY>和<FORM>元素（这些元素必须位于宿主页中）。

### 5.5.1 建立用户控件

Web 用户控件与 Web 窗体页非常相似，它们是使用相同的技术创建的。创建 Web 用户控件，便创建了一个可重复使用的 UI 组件，可以在其他 Web 窗体页上使用该组件。

在 ASP.NET 中，创建用户控件的具体步骤如下。

(1) 新建或打开“项目”。

(2) 在“解决方案管理器”中右击项目名，选择“添加新项”，弹出“添加新项”对话



框，如图 5.28 所示。

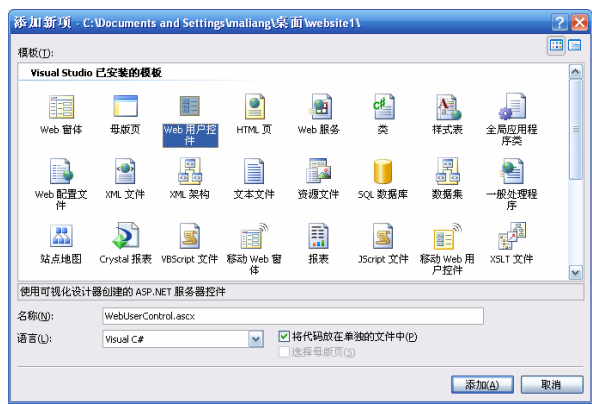


图 5.28 “添加新项”对话框

(3) 在“添加新项”对话框的模板中选择“Web 用户控件”，在“名称”文本框中输入用户控件的文件名（扩展名必须是.ascx），然后按下【添加】按钮。

(4) 在设计器中，像设计普通 Web 页一样设计用户界面和编写事件处理代码。

5.5.2 使用用户控件

可以将 Web 用户控件添加到 Web 窗体页的“设计”视图中。Web 窗体设计器会自动向 Web 窗体页添加该控件的@Register 指令和标记。从此时开始，该控件就成为页的一部分，并将在处理该页时呈现出来。此外，该控件的公共属性、事件和方法将向 Web 窗体页公开并且可以通过编程来使用。

向 Web 窗体页添加用户控件的步骤如下。

- (1) 在 Web 窗体设计器中，打开要将该控件添加到的 Web 窗体页，并确保该页以“设计”视图显示。
- (2) 在解决方案资源管理器中选择用户控件的文件，并将其拖放到 Web 窗体页上。
- (3) 将用户控件添加到 Web 窗体页后，可以像使用其他普通 Web 服务器控件一样对其操作。

**【例 5-11】** 本例创建一个用户控件，并在 Web 页中使用。它的功能是，向用户显示个性化的欢迎信息。

(1) 在网站“Chapter5”中，右击“解决方案资源管理器”内的项目名称，选择“添加新项”，在出现的对话框中选择新建“Web 用户控件”，并输入文件名“welcome.ascx”，单击【确定】按钮。

(2) 该 Web 窗体页中用到 2 个 Label 控件、1 个 TextBox 控件和 1 个 Button 控件，位置见图 5.29。各控件的主要属性设置见表 5.12。

表 5.12 控件的主要属性设置

控 件 名	控件 ID 标识	属 性	属 性 值	备 注
Label	Label1	text	请输入你的姓名：	
Label	Label2	text	空	用于显示信息

TextBox	name	text	空	用于输入用户名
Button	Button1	text	确定	

(3) 为 Button1 的 Click 事件添加处理代码：

```
Label2.Text = "你好！" + name.Text + "，欢迎使用 ASP.NET!" ;
```

(4) 在网站“Chapter”中添加一个命名为“example5-11”的网页，打开“example5-11.aspx”页面，并切换到设计视图，在“解决方案管理器”中把“welcome.ascx”拖放到此页面中。

(5) 按【Ctrl+F5】组合键运行此网页并对其进行测试，运行效果如图 5.29 所示。

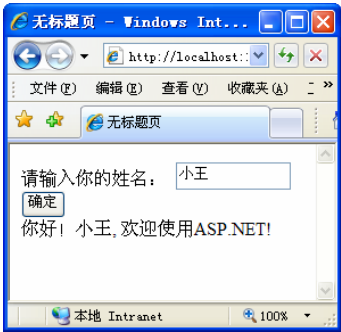


图 5.29 example5-11 页面的运行效果

习 题

- 1. 什么是 Web 服务器控件？它能完成什么功能？
- 2. 如何使用 Button 控件的 Command 事件，响应用户的按钮单击动作？
- 3. 如何使用 CheckBox 和 RadioButton 自动响应用户的选择动作？
- 4. 如何使用 HyperLink 和 LinkButton，用超链接的形式接受用户的单击动作？
- 5. Image 和 ImageButton 的什么属性可以设置其显示的图像？
- 6. TextBox 如何接受密码形式、多行形式的用户输入？
- 7. 如何自动响应 DrowDownList 和 ListBox 中用户的选择动作？
- 8. 面板 Panel 有何作用？
- 9. 如何在网页上，利用 AdRotator 自动生成一个广告栏？
- 10. ASP.NET 的验证控件包括哪些？
- 11. 如何使用必填验证控件，保证用户必须输入某项？
- 12. 如何使用比较验证控件，保证用户的输入是特定的数据类型，或是某个常数，或与另外一个控件中的值具有相同（大于、小于等）关系？
- 13. 如何使用范围验证控件，保证用户的输入在某个范围之内？
- 14. 如何使用正则表达式验证控件，保证用户的输入满足某个构成模式？
- 15. 如何使用自定义验证控件，完成更为灵活的用户输入验证？
- 16. 如何使用验证摘要控件，在页面上统一位置，或者利用弹出对话框输出错误信息？

## 第6章 网站设计

创建 Web 应用程序除了要会正确熟练地使用 ASP.NET 3.5 服务器端控件之外,还要精心设计网站,页面的实际设计和布局同样重要。本章就网站设计涉及的问题进行介绍。

### 6.1 母版页和内容页

在实际的 Web 项目开发中,大多数页面都有些部分是相同的、重复的。如果按照传统的方式则需要为每个页面进行重复设计,不仅浪费时间,而且日后维护也比较麻烦,需要修改多个页面。

母版页是一种很好地解决如上问题的手段。母版页是使用比较多的页面模板之一,也是进行专业级 Web 开发必须掌握的基本技术。

#### 6.1.1 母版页和内容页概述

母版页是指其他网页可以将其作为模板来引用的特殊页面。母版页的扩展名为.master。在母版页中,界面被分成公用区和可编辑区,公用区是多个内容页中共同的区域,可编辑区是内容页中可以编辑的区域。公用区的设计方法与一般页面的设计方式相同,可编辑区通过 ContentPlaceHolder 控件为内容页预留出来,ContentPlaceHolder 内部不能进行设计。一个母版页中可以有一个可编辑区,也可以有多个可编辑区。

母版页代码和普通的.aspx 文件代码格式很相近,最关键的不同是母版页由特殊的 @ Master 指令识别,该指令替换了用于普通.aspx 页的 @ Page 指令,格式如下:

```
<% @ Master Language="C#" CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

可以看出,其实母版页和普通的.aspx 页面非常类似。上述代码指定了母版页的以下几个属性。

- (1) Master Language: 使用的编程语言。
- (2) CodeFile: 母版页的后台代码。
- (3) Inherits: 母版页对应的一个类。

引用母版页的 Web 窗体页面称为内容页,在内容页中,母版页中 ContentPlaceHolder 控件预留的可编辑区会自动替换为 Content 控件,开发人员只需要在 Content 控件区域中填写内容页中不同的内容即可。母版页和内容页的结构关系如图 6.1 所示。

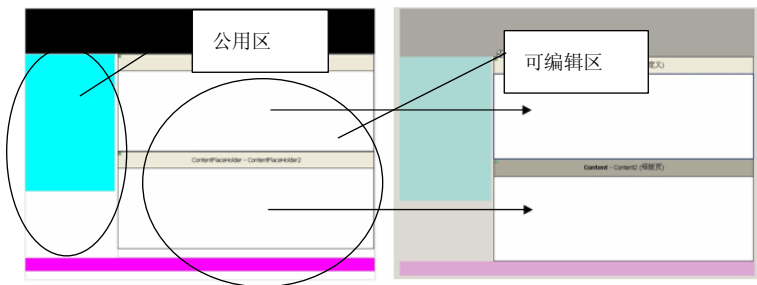


图 6.1 母版页和内容页的结构关系

每个网页的共同部分都可以在母版设计中体现出来。常见的母版页代码结构如下：

```
<%@ Master Language="C#" %> <html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server" >
  <title>Master page title</title>
</head>
<body>
<form id="form1" runat="server">
  <asp:contentplaceholder id="Main" runat = "server" />
  <asp:contentplaceholder id="Footer" runat = "server" />
</form>
</body>
</html>
```

上述代码中的母版页包含两个 ContentPlaceHolder 控件，ID 分别为 Main 和 Footer，它们用于占位。在内容页中，需要创建两个 Content 控件，一个映射到 Main，另一个则映射到 Footer。当客户端请求内容页时，其内容 Content 与母版页的一个副本合并，把定义在 Content 中的特定内容放到 Master 页面的指定占位符处。然后把整个包传递给浏览器，如图 6.2 所示。从用户角度来看，合并后的母版页和内容页是一个完整的页面，且其 URL 访问路径与内容页路径相同。从编程角度来看，这两个页是其各自控件的独立容器。

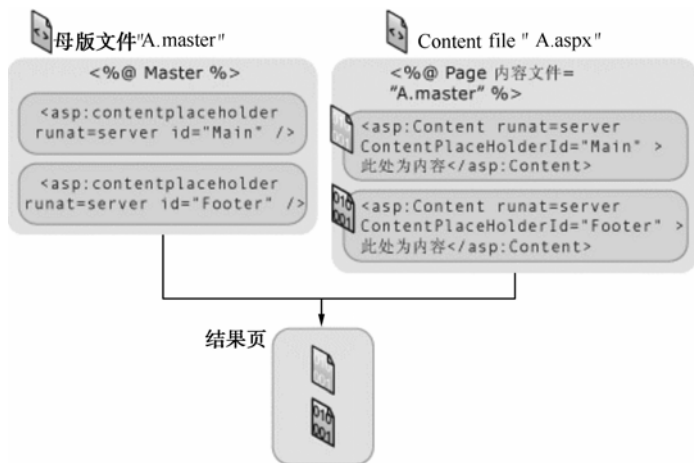


图 6.2 母版页和内容页的代码关系

使用母版页的好处是，开发者可以统一管理和定义页面，使多个页面具有相同的布局风

格，给网页的设计和修改带来了很大的方便。

## 6.1.2 创建母版页和内容页

### 1. 创建母版页

创建母版页的方法是：在 VS 2008 的【解决方案资源管理器】中右击网站，选择“添加新项”，在对话框中选择“母版页”。默认情况下，新建的 Master 页面的默认名称是 MasterPage.master，它位于站点的根目录，如图 6.3 所示。

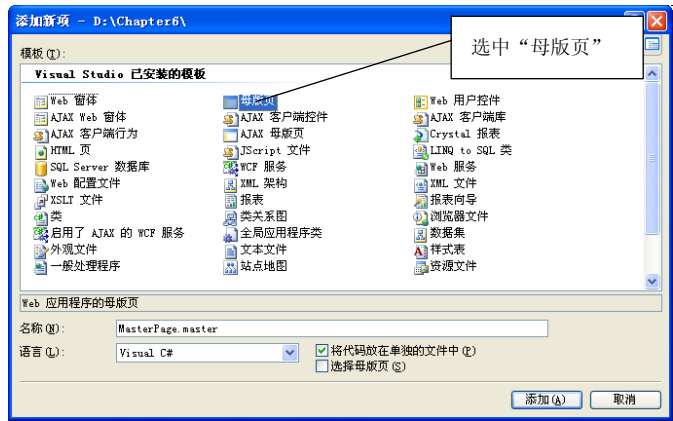


图 6.3 创建母版页

以“.master”为后缀的文件都是母版页，Master 页面可以作为容纳其他页面的容器。每个 Master 页面必须包含下面的元素：

- (1) 基本的 HTML 和 XML 等 Web 标记；
- (2) 位于第一行的`<% @ master ... %>`；
- (3) 带有 ID 的`<asp:ContentPlaceHolder>`标记。

### 2. 创建内容页

创建母版页后，接下来创建内容页。内容页实际上是普通的.aspx 文件，包含除母版页以外的其他非公共部分。

对于内容页有两个概念需要强调：一是内容页中的所有内容必须包含在 Content 控件中；二是内容页必须绑定母版页。

简单地说，内容页应具有下列三个特点：

- ① 没有`<!DOCTYPE HTML...>`和`<html xmlns...>`标记，也没有`<html>`、`<body>`等 Web 元素，这些元素都被放置在母版页；
- ② 在代码的第一行应用`<% @ page MasterPageFile= ... %>`声明所绑定的母版页；
- ③ 包含`<asp:content>`控件。

创建内容页有两种方法。

- (1) 在所要继承的母版页任意位置单击右键，选择“添加内容页”，如图 6.4 所示，就会出现默认的以“Default+序号”命名的内容页的.aspx 文件。
- (2) 在“解决方案资源管理器”上单击右键，新建选项，选择 Web 窗体，并选中“选择

母版页”复选框，如图 6.5 所示。然后在“选择母版页”对话框中选择相应的母版页。创建内容页后，放置在母版页中的控件等内容为灰色，不可编辑。能编辑的部分是在母版页放置的占位符 ContentPlaceHolder 内容区域。在内容页中它表现为 Content 控件，内容页中的控件放置在<asp:content>控件中。内容页中的 Content 控件与母版中的内容区域 ContentPlaceHolder 一一对应，它们通过 ContentPlaceHolder 控件的 ID 属性绑定。

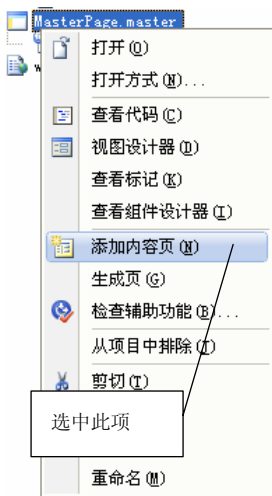


图 6.4 添加内容页

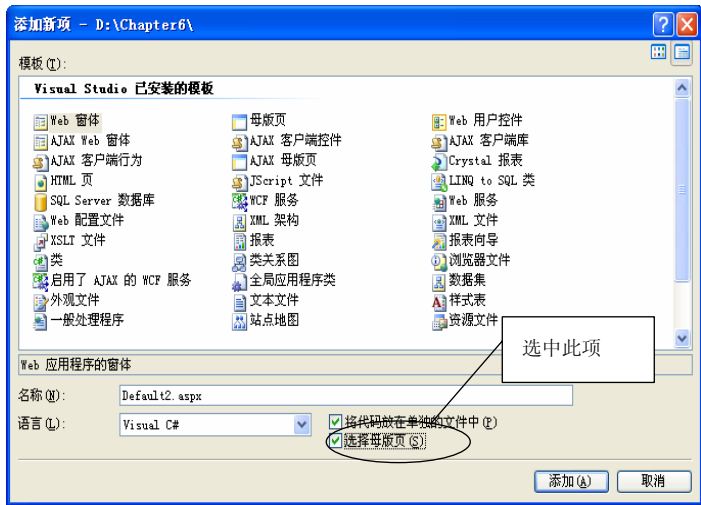


图 6.5 选择母版页

**【例 6-1】** 新建一个网站“Chapter6”，在此网站中添加一个母版页和内容页。

(1) 打开 VS 2008，新建一个网站“Chapter6”，打开“解决方案资源管理器”→右击“Chapter6”→选择“添加新项”选项，在“添加新项”对话框中选择“母版页”→单击“添加”按钮。

(2) 切换到母版页“MasterPage.master”的设计视图，在页面中添加 2 行 2 列的表格，第一行合并单元格，放入 1 个 Image 控件，第二行右边存放 ID 为 ContentPlaceHolder1 的 ContentPlaceHolder 控件。

(3)在网站中新建 1 个命名为“Image”的文件夹，在此文件夹中添加 1 个命名为“ASP.NET”的图片。页面中的ImageUrl 属性设置为“~/Image/ASP.NET.JPG”。从工具箱中拖曳 1 个 Calendar 控件到表格的第二行左列中，如图 6.6 所示。

(4) 打开“解决方案资源管理器”→右击“MasterPage.master”→选择“添加内容页”选项。此时添加了 1 个命名为“Default2.aspx”的内容页，切换到此页面的设计视图，在 ContentPlaceHolder 控件中输入一些文字。

(5) 按【Ctrl+F5】组合键运行网页，结果如图 6.7 所示。

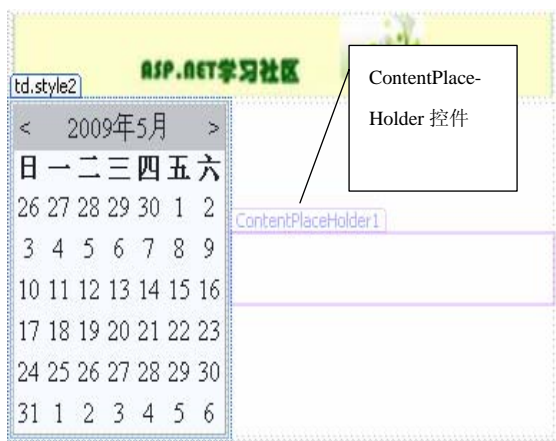


图 6.6 母版页的设计

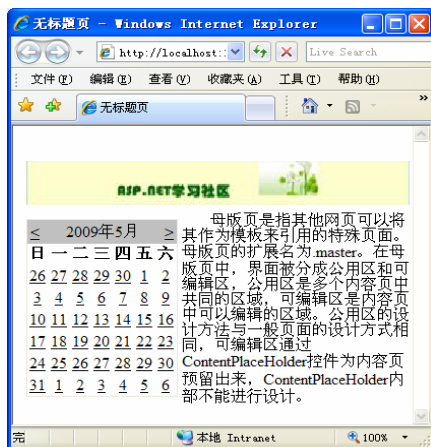


图 6.7 内容页的运行结果

### 6.1.3 访问母版页控件和属性

在内容页面中使用 Master 页面时，可以访问 Master 页面上的控件和属性。Master 页面由内容页面继承时，会提供一个 Master 属性。使用这个属性可以获取 Master 页面中包含的控件值或定制属性。

#### 1. 使用FindControl方法

根据对象的唯一名称，通过使用 Master 页面提供的 MasterPage.FindControl()方法查找要访问的对象，从内容页面上访问母版页面上的控件。

例如，在母版页面上有一个ID为Label1的标签控件，在内容页面中有一个ID为LabelTitle的标签控件，在页面加载事件中，让内容页面的控件LabelTitle获取母版页面控件Label1的Text，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    LabelTitle.Text = (Master.FindControl("Label1") as Label).Text;
}
```

当然，这种交互打破了基于类设计和封装的原则。如果内容页修改其他页面的内部逻辑，则编辑母版页时很可能破坏了这种依赖关系，从而产生脆弱的代码模型。

#### 2. 使用MasterType指令

另一个能够访问母版页面的方法是在内容页面中加入 MasterType 指令。在这个指令里指定.master文件的虚拟路径。

下面是添加了 MasterType 指令的某个内容页的前两行代码：

```
<%@ Page Language="C#" MasterPageFile="~/MyMaster1.master" %>
<%@ MasterType VirtualPath="~/MyMaster1.master" %>
```

MasterType 指令允许对 Master 页面进行强类型化引用，通过 Master 对象访问 Master 页面的属性。

**【例 6-2】 访问母版页。**

(1) 打开 VS 2008，在【例 6-1】创建的网站 Chapter6 上添加一个 Web 窗体，并选择母版，具体方法在前面已经讲述。选择母版 Master.master，给新的 Web 窗体命名为 example6-2.aspx。

(2) 切换到内容页 example6-2.aspx 的设计视图下，在 ContentPlaceHolder 控件中添加 1 个 ID 为 Label1 的 Label 控件，在 example6-2.aspx.cs 文件中添加代码，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = (Master.FindControl("Calendar1") as Calendar).Today.ToString();
}
```

(3) 按【Ctrl+F5】组合键运行网页，结果如图 6.8 所示。

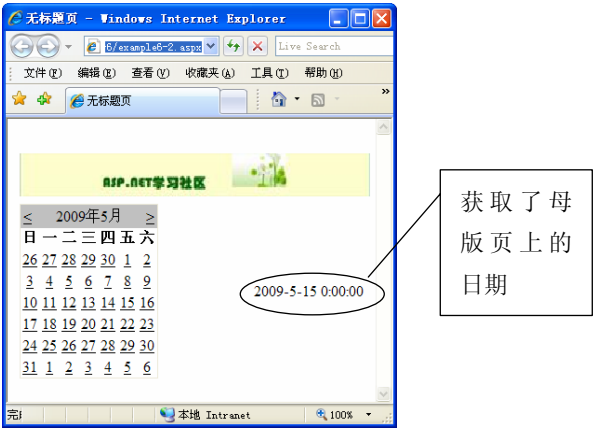


图 6.8 example6-2 网页运行的结果

**6.2 主题和皮肤**

应用“主题和皮肤”功能可以轻松地实现对网站美观的控制。可以将样式和布局信息分解为单独的文件组，统称为“主题”。主题可应用于任何站点，影响站点中每一个网页和控件的外观。通过更改主题即可轻松地维护对站点的样式更改。用户还可以与其他开发人员共享主题，利用主题方便地控制页面外观。把所有与页面外观有关的控制文件和资源文件放在主题目录中，页面只需切换主题，则主题目录下所有的控制文件和资源文件就会自动切换。

**6.2.1 主题概述**

主题是 ASP.NET 3.5 基于文本的样式定义。多个主题的优点在于，设计站点时可以不考虑样式，以后应用样式时也无须更新页或应用程序代码。此外，还可以从外部源获得自定义主题，以便将样式设置应用于应用程序。可以在应用程序、页面或服务器控件级别上应用 ASP.NET 中的主题。

**1. 主题与CSS**

CSS 规则只限于一组固定的样式特性。它们允许重用特定的格式化细节（字体、边框、前景和背景色等），但它们显然不能控制 ASP.NET 控件的其他方面。例如，CheckBoxList 控



件有一些用于控制如何把项目组织为行或列的属性。此外，如果希望在定义控件格式的同时定义控件的部分行为，则不可能通过 CSS 实现。

主题弥补了中间的鸿沟。和 CSS 相似，主题允许定义一组作用于多个页面中的控件的样式特性。不过，和 CSS 不同的是，主题不是由浏览器实现。相反，它们是在服务器端实现的 ASP.NET 自有的解决方案。虽然主题不会代替样式，但它们可以提供一些 CSS 不能提供的特性。以下是它们的主要区别。

(1) 主题基于控件而不是 HTML。所以，主题允许定义和重用几乎所有的控件属性。主题应用在服务器上。当主题作用到页面时，格式化后的最终页面被传送给用户。而使用样式表时，浏览器同时接收到页面和样式信息并在客户端合并它们。

(2) 可以通过配置文件来应用主题。这样不必修改任何一个页面就可以对整个文件或整个网站应用主题。

(3) 主题不会像 CSS 那样级联。如果在一个主题和一个控件中同时指定了一个属性，那么主题中定义的值会覆盖控件的属性。倘若要改变这个行为，则可以提高页面属性的优先级，这样主题的行为将更像样式表。

## 2. 皮肤文件

CSS 不能对服务器控件的样式进行设置，因此 ASP.NET 中提出了皮肤文件的概念。

每个主题文件夹下都可以包含一个或多个皮肤文件。各种控件的样式在主题中被指定为皮肤 (Skin)。皮肤文件又称外观文件。外观文件是一个用来描述 Web 服务器控件外观属性的设置集合。一个 .skin 文件可以包含一个或多个控件类型的外观。可以为每个控件在单独的文件中定义外观 (例如，单独为日历控件定义日历控件的皮肤)，也可以在一个文件中定义所有主题的外观 (将所有控件外观的定义都集中在一个皮肤文件中)。已命名外观是设置了 SkinID 属性的控件外观，通过设置控件的 SkinID 属性将已命名外观显式应用于控件。通过创建已命名外观，可以为应用程序中同一控件的不同实例设置不同的外观。

下面是一个外观文件中一个文本框服务器控件的外观设置代码：

```
<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Verdana"Font-Size="X-Small"
BorderStyle="Dotted" BorderWidth="5px" BorderColor="#000000" Font-Bold="False"
SkinID="TextboxDotted" />
```

创建皮肤文件时，需要注意：

- (1) 要使用 .skin 扩展文件名；
- (2) 添加常规控件定义 (使用声明性语法) 时，仅可以包含要为主题设置的属性而不包括 ID 属性；
- (3) 控件定义必须包含 runat="server" 属性。

## 3. 主题的类型

主题有两种类型，一种是页面主题，另一种是全局主题。

(1) 页面主题：仅应用于单个 Web 应用程序，也称为应用程序主题。应用程序主题是指在 Web 应用程序的 App\_Themes 文件夹下的一个或多个特殊文件夹，主题的名称是文件夹的名称。每个主题都是一个主题文件夹，其中包含控件外观、样式表、图形文件和其他资源，该文件夹是作为网站中的 \App\_Themes 文件夹的子文件夹创建的。每个主题都是 \App\_Themes

文件夹的一个不同的子文件夹。图 6.9 展示的是一个典型的页面主题，它定义了五个主题，不同的主题下有着不同的组成。

(2) 全局主题：可以应用于服务器上的所有网站的主题。当维护同一个服务器上的多个网站时，可以使用全局主题定义其整体外观。

#### 4. 文件存储和组织方式

主题是指页面和控件外观属性设置的集合，由 ASP.NET 支持的具有特殊含义的文件夹构成。在主题中，可以包含各种页面外观控制文件和资源文件，主要有外观文件(扩展名为.skin)、级联样式表文件(扩展名为.css)、脚本文件(扩展名为.js)、资源文件(扩展名为.resx)、图像文件、声音文件等。主题是在网站或 Web 服务器上的特殊目录中定义的。

在默认情况下，主题存储在网站中的 App\_Themes 目录下；皮肤则存储于相应主题文件夹中，是以.skin 为后缀的外观文件。

如图 6.10 所示，在 App\_Themes 文件夹下包括了两个子文件夹 Blue 和 Red，表示应用程序中定义了两个主题。每个主题文件夹下都可以包含一个或多个外观文件。在应用程序主题较多、页面内容较复杂的情况下，必须将外观文件组织好。

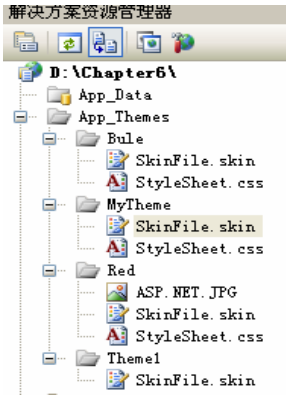


图 6.9 页面主题目录结构图

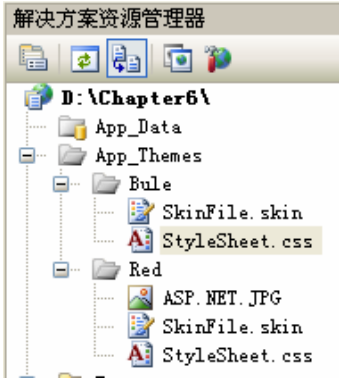


图 6.10 外观文件组织方式一

常见的外观文件组织方式如下。

(1) 根据 SkinID 组织：每个外观文件都包含具有相同 SkinID 的多个控件外观定义，见图 6.10。这种方式适用于站点页面较多、设置内容复杂的情况。

(2) 根据控件类型组织：每个外观文件都包含特定控件的一组外观定义，如图 6.11 所示。这种方式适用于站点页面中包含控件较少的情况。

(3) 根据网页文件名组织：每个外观文件定义某一个网页文件页面中控件的外观，站点中有多少个网页文件，就有对应数目的外观文件，如图 6.12 所示。站点中包含页面较少时可以采用这种方式。



图 6.11 外观文件组织方式二



图 6.12 外观文件组织方式三

## 6.2.2 创建主题

在 ASP.NET 3.5 中创建主题的过程比较简单，但要创建出美观实用的主题还是需要一些艺术细胞的。下面用一个例子讲述创建主题的方法和步骤。

### 1. 创建主题文件夹

打开“解决方案资源管理器”→右击项目名→选择“添加”→“添加 ASP.NET 文件夹”→“主题”。如果不存在 App\_Themes 文件夹，则系统自动创建它，并在该文件夹下添加一个主题；如果已经存在该文件夹，就直接在该文件夹下添加新的主题。

### 2. 创建外观文件

成功创建主题文件夹后，就可以在创建的主题目录下添加外观控制文件和资源文件。

在编写外观文件（.skin）时，系统并没有提供控件属性设置的智能化提示功能。所以一般不在外观文件中直接编写代码定义控件的外观，而是先在 Web 窗体文件中拖放控件并设置控件的属性，然后将自动生成的代码复制到外观文件中，再进行修改。步骤如下。

（1）右击主题文件夹，选择“添加新项”，添加一个皮肤文件“SkinFile.skin”。

（2）在网站内添加一个临时的 Web 窗体文件 Temp.aspx，在设计视图下，将需要设置外观的控件拖放到页面中，最好一行放置一个控件，以方便代码查看。

（3）利用属性窗口及可视化功能对控件进行配置，设置控件的背景色、前景色、字体等外观属性。

（4）将相应控件的源代码复制到外观文件“SkinFile.skin”中，并去掉控件的 ID 属性。

（5）如果源代码中同种控件出现多个，则需给控件添加不同的 SkinID 属性。当 Web 窗体页面内的控件的 SkinID 属性值与某 SkinID 属性值相同时，就会采用此外观效果。

一个定义好的包含多个 Skin 选项的外观文件代码片段如下：

```
<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"
    Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
    BorderColor="#004000" Font-Bold="True" />
<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Verdana"
    Font-Size="X-Small" BorderStyle="Dotted" BorderWidth="5px"
    BorderColor="#000000" Font-Bold="False" SkinID="TextboxDotted" />
<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Arial"
    Font-Size="X-Large" BorderStyle="Dashed" BorderWidth="3px"
    BorderColor="#000000" Font-Bold="False" SkinID="TextboxDashed" />
```

### 3. 添加CSS文件

除了在.skin 文件中创建的服务器控件定义之外，还可以使用 CSS 进行进一步的定义，使 HTML 服务器控件、HTML 和原始的文本都根据主题来改变。VS 2008 提供了样式构建器，使用此工具可以轻松地为主题创建 CSS 文件。

（1）右击主题文件夹，选择“添加新项”，添加一个样式表文件“StyleSheet.css”。添加后的文件默认有一个空的 body 样式规则，其代码如下：

```
body
{ }
```

(2) 右击空白处，选择“添加样式规则”，如图 6.13 所示。弹出“添加样式规则”对话框。如图 6.14 所示，在其中选择要设置样式的元素、类名等，可以添加新的样式规则。新生成的样式规则为元素名 {}。



图 6.13 添加样式规则

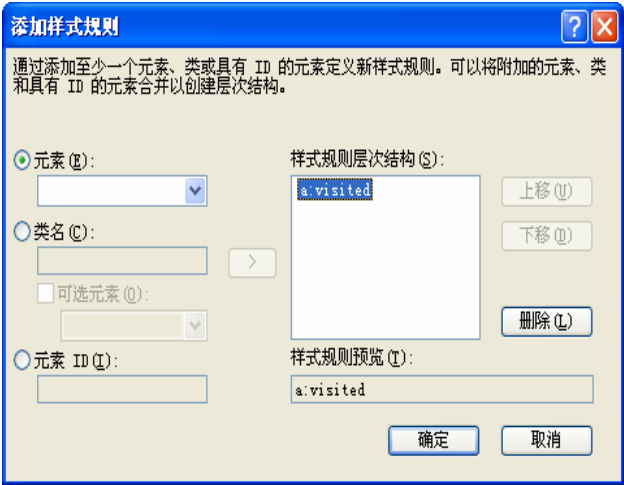


图 6.14 “添加样式规则”对话框

(3) 在生成的样式规则的 HTML 标签对应的花括号内，右击空白处，选择“生成样式”，如图 6.15 所示。弹出“修改样式”对话框，如图 6.16 所示。“修改样式”对话框可以用来定义或修改元素的样式。若是首次使用，则为定义样式，再次使用则为修改样式。在其中选择要设置的字体、背景等进行设置，就会自动生成对应的属性和值。可见，VS 2008 中的对话框可以完全定义 CSS 页面，无须编码。

下面是一个定义好的 CSS 文件代码片段：

```
body
{ font-size: x-small;
  font-family: Verdana;
  color: #004000;
}
A:link
{ color: Blue;
  text-decoration: none;
}
```



图 6.15 生成样式

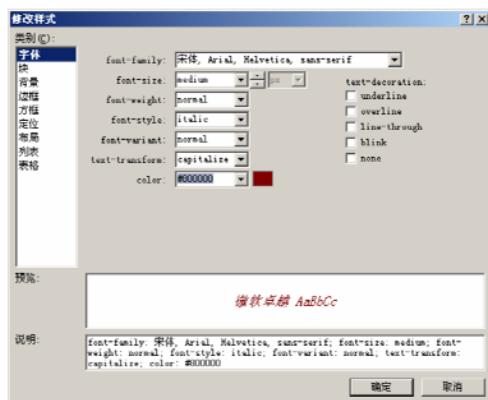


图 6.16 “修改样式”对话框

CSS 文件是一个解释性的文件，样式在.css 文件中的定义顺序非常重要。一些样式会改变前面的样式，所以应确保样式定义的顺序正确。在和皮肤文件定义的外观属性冲突时，.skin 文件将优先于.css 文件的样式。

#### 4. 给主题添加图像

主题可以将图像合并到样式定义中，许多控件都可以使用图像创建更美好的外观。把图像合并到统一使用主题的服务器控件中，需要在主题文件夹中创建 **Images** 文件夹。使用 **Images** 文件夹中的文件有两个简单的方法。

(1) 把图像直接合并到.skin 文件中，示例代码片段如下。

```
<asp:TreeView runat="server" ...
    LeafNodeStyle-ImageUrl="images\summer_iconlevel.gif"
    RootNodeStyle-ImageUrl="images\summer_iconmain.gif"
    ...
</asp:TreeView>
```

(2) 如果服务器控件并没有 **image** 属性，那么就必须使用.skin 文件和 CSS 文件。此时可以将基于主题的图像放在任意元素上。

下面的代码是在 CSS 文件中的 CSS 类中给元素定义使用背景图像：

```
.theme_header { background-image :url( images/smokeandglass_brownfadetop.gif); }
.theme_highlighted { background-image :url( images/smokeandglass_blueandwhitef.gif); }
.theme_fadeblue { background-image :url( images/smokeandglass_fadeblue.gif); }
```

在 CSS 文件中定义过 CSS 类之后，就可以在定义服务器控件时，在.skin 文件中使用它们。代码如下：

```
<asp:Calendar runat="server" BorderStyle="double" BorderColor="#E7E5DB"
BorderWidth="2" BackColor="#F8F7F4" Font-Size=".9em" Font-Names="Verdana">
<TodayDayStyle BackColor="#F8F7F4" BorderWidth="1" BorderColor="#585880" ForeColor=
"#585880" /><OtherMonthDayStyle BackColor="transparent" ForeColor="#CCCCCC" />
<SelectedDayStyleForeColor="#6464FE" BackColor="transparent" CssClass="theme_highlighted" />
<TitleStyle Font-Bold="True" BackColor="#CCCCCC" ForeColor="#585880"
```

```
BorderColor="#CCCCCC" BorderWidth="1pt" CssClass="theme_header" />
...
</asp:Calendar>
```

## 6.2.3 应用主题

### 1. 给单个页面应用主题

定义页面主题后，可以使用 `@ Page` 指令的 `Theme` 属性将该主题放置在单个页面上。例如：

```
<% @ Page Language="C#" Theme="ThemeName" %>
```

还可以使用 `@ Page` 指令的 `StyleSheetTheme` 属性将该主题放置在单个页面上。例如：

```
<% @ Page Language="C#" StyleSheetTheme="ThemeName" %>
```

可以手工做这个修改，在设计时打开 `.aspx` 文件，切换到设计视图，右击选择“属性”，在属性窗口顶部的下拉列表中选择“Document”，在列表中定位到“`StyleSheetTheme`”，设置其值为某个主题名称，本页面就会自动套用主题内的外观控制文件和资源文件。

### 2. 禁用主题

默认情况下，主题将重写页和控件外观的本地设置。当控件或页已经有预定义的外观而又不希望主题重写它时，可以禁用此行为。对于页面，禁用主题的方法为：

```
<% @ Page EnableTheming="false" %>
```

对于控件，禁用主题的方法为：

```
<asp:Calendar id="Calendar1" runat="server" EnableTheming="false" />
```

#### 【例 6-3】 创建和应用主题示例。

(1) 运行 VS 2008，在资源管理器中右击网站“Chapter6”→“添加 ASP.NET 文件夹”→“主题”，系统会自动创建 `App_Themes` 目录，并在其下创建“主题 1”目录。

(2) 右击“主题 1”→选择“添加新项”，在出现的窗口中选择“外观文件”，用同样的方法在主题内添加一个样式表文件，其目录结构如图 6.17 所示。

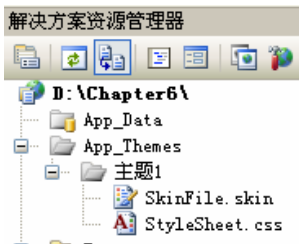


图 6.17 主题目录

(3) 添加一个 Web 窗体文件“Temp.aspx”，切换到设计视图，在页面上拖放两个 `Label`

控件。按表 6.1 所示分别设置两个 Label 控件的外观属性。

表 6.1 控件外观属性的设置

控 件 ID	属 性 设 置
Label1	BackColor: #0099CC, ForeColor: #FF0066
Label2	BackColor: #33CC33; ForeColor: #660033; BorderStyle="Dotted"

(4) 切换到“Temp.aspx”的源代码视图，把控件的代码复制到“SkinFile.skin”文件中，并去掉相关控件的 ID 属性，在第一个 Label 中添加 SkinID="other"，修改成如下的代码：

```
<asp:Label SkinID="other" runat="server" BackColor="#0099CC" ForeColor="#FF0066"></asp:Label>
<asp:Label runat="server" BackColor="#33CC33" ForeColor="#660033" BorderStyle="Dotted">
</asp:Label>
```

(5) 在“StyleSheet.css”的文件中加入如下的样式规则：

```
body
{
    text-align: center;
    background-repeat: no-repeat;
    font-size: larger;
}
```

(6) 添加一个 example6-3.aspx 的文件，切换到设计视图，打开“属性”窗口，在属性窗口顶部的下拉列表中选择“Document”，在列表中定位到“StyleSheetTheme”，设置其值为“主题 1”。

(7) 向“example6-3.aspx”文件中拖放两个 Label 控件，会发现两个控件的显示效果如图 6.18 所示。

(8) 修改 Label2 的 SkinID 属性为“other”，则 Label2 会自动换成如图 6.19 所示的外观。



图 6.18 应用主题 1 的默认外观



图 6.19 根据 SkinID 应用外观

## 6.3 网站导航

虽然使用超链接或者服务器代码可以从一个网页切换到另一个网页，但是，当网站中的页面很多时，页面之间的层次关系将会变得很复杂。ASP.NET 站点导航功能则提供了方便的导航方法。

站点导航主要提供了如下功能。

(1) 使用站点地图描述站点的逻辑结构，通过 SiteMapPath 可以自动从地图文件中获取文件所在的层次，自动生成导航栏。

(2) 提供导航控件，在页面上显示导航菜单。

在 VS 2008 中，提供的导航控件有 SiteMapPath 控件、Menu 控件和 TreeView 控件。一般情况下，开发人员利用站点地图和 SiteMapPath 控件实现自动导航，利用 Menu 控件或者 TreeView 控件实现自定义导航。

### 6.3.1 站点地图和SiteMapPath控件

#### 1. 站点地图

站点地图文件是用来描述站点逻辑结构的 XML 文件，该文件必须保存在 Web 应用程序的根目录下才起作用，站点地图文件的扩展名为.sitemap。例如：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="default.aspx" title="ASP.NET3.5 教程" description="主页">
    <siteMapNode url="WebControls.aspx" title="Web 服务器控件" description=" Web 服务器控件" >
      <siteMapNode url="example4-2.aspx" title="文本框" />
      <siteMapNode url="example4-3.aspx" title="标签" />
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

该文件中只能有一个<siteMap>元素。在这个<siteMap>元素中，有一个<siteMapNode>元素。这一般是应用程序的起始页面。在上述文件代码中，根<siteMapNode>指向起始页面 default.aspx。

表 6.2 描述了<siteMapNode>元素中最常见的一些属性。

表 6.2 <siteMapNode>元素的常见属性

属 性	说 明
title	title 属性提供链接的文本描述。这里使用的 String 值是用于链接的文本
description	description 属性不仅说明该链接的作用，而且还用于链接上的 ToolTip 属性。ToolTip 属性是终端用户把光标停留在链接上几秒后显示出来的黄框



属 性	说 明
url	url 属性描述了文件在解决方案中的位置。如果文件在根目录下，就使用文件名。如果文件位于子文件夹下，就在这个属性的 String 值中包含该文件夹，例如，“MySubFolder/Markets.aspx”

2. SiteMapPath控件

SiteMapPath 控件以导航路径的方式显示当前页在站点中的位置，定义好站点地图以后，只需要将该控件拖放到站点地图中定义过的.aspx 页面上，它就会自动实现导航，不需要开发者编写任何代码。

注意：只有包含在站点地图中的网页才能被 SiteMapPath 控件导航；如果将 SiteMapPath 控件放置在站点地图中未列出的网页中，则该控件将不会显示任何信息。

SiteMapPath 控件的常用属性如表 6.3 所示。

表 6.3 SiteMapPath 控件的常用属性

属 性	说 明
CurrentNodeStyle	定义当前节点的外观样式
NodeStyle	定义 SiteMapPath 中所有导航节点的外观样式
PathSeparator	设置导航路径中节点之间的分隔符
PathSeparatorStyle	定义分隔符的样式
RootNodeStyle	定义根节点样式

【例 6-4】 在网站 Chapter6 中，利用 SiteMapPath 控件实现自动导航。

- (1) 运行 VS 2008→打开“解决方案资源管理器”→右击“Chapter6”→“添加新项”→选择“站点地图”模板→单击“添加”按钮，如图 6.20 所示。
- (2) 将 Web.config 文件中的内容改成如下，保存文件，完成站点地图的设计。

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="Default.aspx" title="首页" description="回到首页">
    <siteMapNode url="example6-2.aspx" title="【例 6-2】" description="" />
    <siteMapNode url="example6-3.aspx" title="【例 6-3】" description="" />
  </siteMapNode>
</siteMap>
```

- (3) 切换到 Default.aspx 的设计视图，向页面内拖放一个 SiteMapPath 控件。同样，在其他页面上拖放 SiteMapPath 控件，都会生成对应的导航路径。
- (4) 运行 example6-3.aspx 页面，结果如图 6.21 所示。

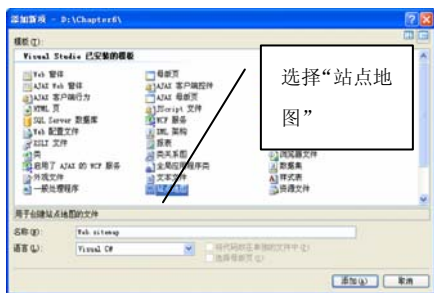


图 6.20 创建站点地图文件图



图 6.21 导航效果图

## 6.3.2 用Menu控件导航

Menu 控件主要用于创建一个页面功能菜单，让用户可以快速地选择不同的页面。该控件可以包括一个主菜单和多个子菜单。菜单有静态和动态两种显示模式。静态显示模式是指定义的菜单始终完全显示，动态显示模式是指需要用户将鼠标停留在菜单项上时才显示子菜单。Menu 控件的常用属性如表 6.4 所示。

表 6.4 Menu 控件的常用属性



属 性	说 明
Orizntation	设置菜单的展开方向。可以取值为 Horizontal 和 Vertical 两个选项
MaximumDynamicDisplayLevels	设置动态菜单的最大层数，默认为 3

Menu 控件的用法非常灵活，设计者可以利用它来定义各种菜单样式，实现类似于 Windows 窗体菜单的功能。

**【例 6-5】** 在网站 Chapter6 中新建 1 个网页，用于体现 Menu 控件的使用。

(1) 在网站 Chapter6 中添加 1 个命名为“example6-5”的网页。

(2) 切换到设计视图下，从工具箱导航控件选项卡中拖放 1 个 Menu 控件到此页面。

(3) 右击 Menu 控件→单击“编辑菜单项”，在“菜单项编辑器”中单击“添加根项”图标，设置属性，Text 为“首页”、NavigateUrl 为“~/Default.aspx”。单击“添加子项”图标，设置属性，Text 为“【例 6-2】”、NavigateUrl 为“~/example6-2.aspx”。再次单击“添加子项”图标，设置属性，Text 为“【例 6-3】”、NavigateUrl 为“~/example6-3.aspx”，如图 6.22 所示。

(4) 按【Ctrl+F5】组合键运行网页，结果如图 6.23 所示。

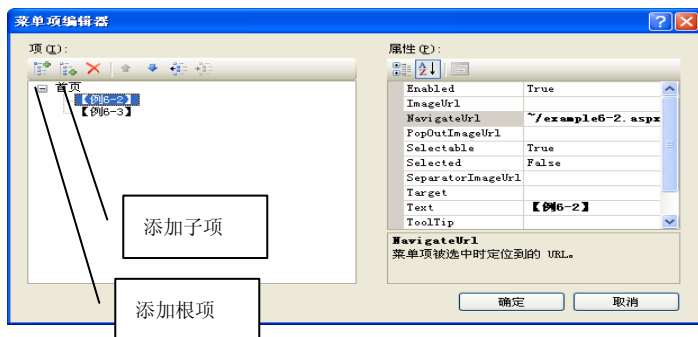


图 6.22 编辑菜单项

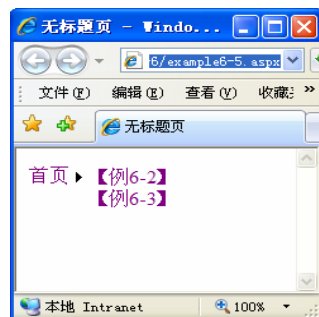


图 6.23 example6-5 网页运行的结果

Menu 控件可以由配置文件显示整个网站的结构，让用户单击不同的链接，从而转到不同的页面。除了配置文件以外，它还要一个 SiteMapDataSource 数据源，这个数据源会自动找到网站地图的配置文件。

**【例 6-6】** 使用 Menu 控件绑定到 SiteMapDataSource 进行导航。

(1) 在网站“Chapter6”中添加一个网页，命名为“example6-6.aspx”。

(2) 打开“example6-6.aspx”页面的设计视图，从“工具箱”中的“数据”选项卡中拖放一个 SiteMapDataSource 控件至设计页面，其 ID 为 SiteMapDataSource1，它会自动配置上例中已创建完成的网站地图文件 Web.sitemap。

(3) 从“工具箱”中的“导航”选项卡中拖放一个 Menu 控件至设计页面，设置 Menu 控件的 DataSourceId 属性为 SiteMapDataSource1。

(4) 按【Ctrl+F5】组合键运行网页，结果和【例 6-5】的运行结果一样，见图 6.23。

在此例中，使用 SiteMapDataSource 控件自动处理应用程序的 Web.sitemap 文件。该示例包含的另一项是 Menu 控件，它使用经典的 ID 和 Runat 属性，并使用 DataSourceID 属性把它链接到从 SiteMapDataSource 控件提取出来的数据上。

### 6.3.3 用TreeView控件导航

TreeView 控件和 Menu 控件具有相同的功能，只是外观不同，TreeView 控件以树形结构显示节点项，根节点下也可以包含多个子节点，子节点又可以包含子节点，最下层是叶节点。TreeView 控件可以使用上面介绍的 Menu 控件的设置方法进行设置，这里就不介绍了，下面介绍用程序添加属性。

TreeView 控件的每个节点是一个 TreeNode 对象，具有 Text 属性和 Value 属性，Text 属性指定在节点显示的文字，Value 属性获取节点的值。每个节点有选定和导航两种状态，NavigateUrl 属性决定节点的状态，当该属性不为空字符串("")值时为导航状态，否则为选择状态。默认情况下，会有一个节点处于选择状态。

**【例 6-7】** 利用 TreeView 控件创建自定义导航。

(1) 在网站“Chapter6”中添加一个名为“example6-7.aspx”的网页，然后切换到设计视图，向网页中拖放一个 TreeView 控件。

(2) 在首次加载页面时，首先创建一个根节点，不带任何导航信息，然后将该节点添加子节点信息。在代码页中添加代码，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        TreeView1.ShowLines = true;           //在控件中显示网格线
        TreeNode rootNode = new TreeNode();    //定义根节点
        rootNode.Text = "网站 Chapter6";
        TreeNode tr1 = new TreeNode();         //定义子节点
        tr1.Text = "【例 6-2】";
        tr1.NavigateUrl = "~/example6-2.aspx";
        rootNode.ChildNodes.Add(tr1);         //把子节点添加到根节点
        TreeNode tr2 = new TreeNode();
```

```

        tr2.Text = "【例 6-3】";
        tr2.NavigateUrl = "~/example6-3.aspx";
        rootNode.ChildNodes.Add(tr2);
        TreeNode tr3 = new TreeNode();
        tr3.Text = "Menu 控件";
        rootNode.ChildNodes.Add(tr3);
        TreeNode tr3_1 = new TreeNode();
        tr3_1.Text = "【例 6-5】";
        tr3_1.NavigateUrl = "~/example6-5.aspx";
        tr3.ChildNodes.Add(tr3_1);           //添加二级子节点
        TreeNode tr3_2 = new TreeNode();
        tr3_2.Text = "【例 6-6】";
        tr3_2.NavigateUrl = "~/example6-6.aspx";
        tr3.ChildNodes.Add(tr3_2);           //添加二级子节点
        TreeView1.Nodes.Add(rootNode);       //把根节点添加到 TreeView 控件中
    }
}

```

(3) 按【Ctrl+F5】组合键运行网页，运行效果如图 6.24 所示。



图 6.24 TreeView 控件运行效果图

TreeView 控件的属性比较丰富, ShowLines 属性确定各节点之间是否显示连线。TreeNode 对象代表 TreeView 控件的一个节点, 该对象的 ChildNodes 属性包含节点的子节点。

## 习 题

1. 母版页有何作用?
2. 如何创建一个母版页, 并利用其使多个页面具有统一的布局?
3. 母版页和内容页是如何融合在一起的?
4. 如何在母版页内部使用控件触发事件?
5. 如何在母版页中访问内容页中的控件?
6. 比较 ASP.NET 3.5 中多种导航方式的优缺点及其使用场合。

# 第 7 章 ASP.NET 数据库编程

本章将介绍数据库的一些基本知识、ASP.NET 3.5 中的数据源控件和数据绑定控件，以及 ADO.NET 数据访问编程模型。

## 7.1 数据库基础

### 7.1.1 数据库和数据库管理系统

#### 1. 数据库

数据库 (Database, DB)，顾名思义，就是存放数据的仓库，其特点是数据按照数据模型组织，是高度结构化的，可供多个用户共享并且具有一定的安全性。

#### 2. 数据库管理系统

数据库管理系统 DBMS (DataBase Management System)，是位于用户应用程序和操作系统之间的数据库管理软件，其主要功能是组织、存储和管理数据，高效地访问和维护数据，即提供数据定义、数据操纵、数据控制和数据维护等功能。

本教材使用的主要是 ASP.NET 操作 SQL Server 2005 数据库。数据库管理器 (SQL Server Management Studio) 如图 7.1 所示 (图中的数据库是后来添加的)。

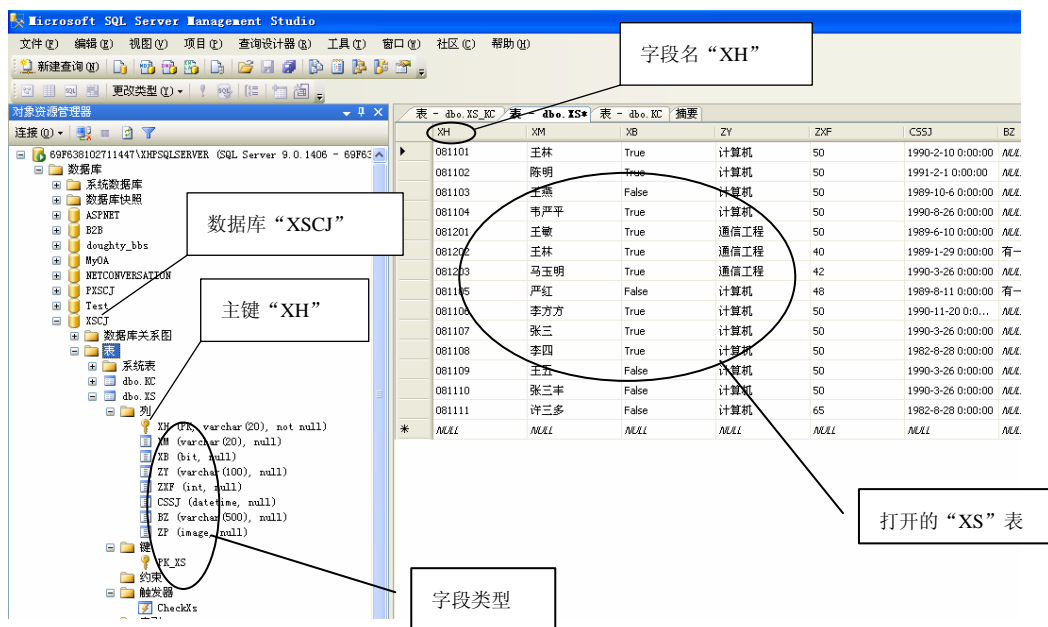


图 7.1 SQL Server Management Studio 部分截图

7.1.2 表和视图

实际开发中使用的数据库几乎都是关系型的，表是关系数据库中最主要的数据库对象，它是用来存储和操作数据的一种逻辑结构。表由行和列组成，因此也称之为二维表。

1. 表（Table）

表是日常工作和生活中经常使用的一种表示数据及其关系的形式，例如，表 7.1 所示就是一个学生情况表。

表 7.1 学生情况表

学 号	姓 名	专 业 名	性 别	出 生 时 间
081101	王 林	计算机	男	1990/10/01
081102	王 巍	计算机	女	1991/02/08
081103	林 滔	通信工程	男	1990/04/06
081104	江为中	通信工程	男	1990/12/08

每个表都有一个名字，以标识该表。例如，表 7.1 中的名字是学生情况表，它共有 5 列，每一列也都有一个名字，描述了学生某一方面的特性。每个表由若干行组成，表的第一行为各列标题，即“栏目信息”，其余各行都是数据。例如，表 7.1 分别描述了 4 位同学的情况。下面是表的定义。

(1) 表结构

每个数据库包含了若干个表。每个表具有一定的结构，称之为“表型”。所谓表型，是指组成表的各列的名称及数据类型，也就是日常表格中的“栏目信息”。

(2) 记录

每个表包含了若干行数据，它们是表的“值”。

(3) 字段

每个记录由若干个数据项构成，将构成记录的每个数据项称为字段（Field）。字段包含的属性有字段名、字段数据类型、字段长度及是否为关键字等，其中字段名是字段的标识，字段的数据类型可以是多样的，如整型、实型、字符型、日期型或二进制型等。

(4) 关键字

在学生情况表中，若不加以限制，则每个记录的姓名、专业名、性别和出生时间这 4 个字段的值都有可能相同，但是学号字段的值对表中所有记录来说一定不同，即通过“学号”字段可以将表中的不同记录区分开来。

若表中记录的某一字段或字段组合能唯一标识记录，则称该字段或字段组合为候选关键字（Candidate key）。若一个表中有多个候选关键字，则选定其中一个为主关键字（Primary key），也称为主键。当一个表仅有唯一的一个候选关键字时，该候选关键字就是主关键字。例如，学生情况表的主关键字为学号。

若某字段或字段组合不是数据库中 A 表的关键字，但它是数据库中 B 表的关键字，则称该字段或字段组合为 A 表的外关键字（Foreign key）。

例如，设学生数据库有三个表，分别为学生表、课程表和学生成绩表，其结构分别为：

学生表（学号，姓名，专业，性别，出生年月）  
课程表（课程号，课程名，学分）  
学生成绩表（学号，课程号，分数）  
（带下划线的字段或字段组合为关键字）

可见，单独的学号、课程号都不是学生成绩表的关键字，但它们分别是学生表和课程表的关键字，所以它们都是学生成绩表的外关键字。

外关键字表示了表之间的参照完整性约束。例如，在学生数据库中，在学生成绩表中出现的学号必须是学生表中出现的，同样课程号也必须是课程表中出现的。若在学生成绩表中出现了一个未在学生表中出现的学号，则会违背参照完整性约束。

## 2. 视图（View）

视图是从一个或多个表（或视图）导出的表。

视图与表不同，它是一个虚表，即视图所对应的数据不进行实际存储，数据库中只存储视图的定义，当对视图的数据进行操作时，系统根据视图的定义去操作与视图相关联的基本表。视图一经定义以后，就可以像表一样被查询、修改、删除和更新。使用视图具有便于数据共享、简化用户权限管理和屏蔽数据库的复杂性等优点。

### 7.1.3 用VS 2008 创建数据库和表

VS 2008 自带有 SQL Server 2005 Express 版（是可选安装的）。

#### 1. 创建数据库

用 VS 2008 创建数据库的步骤如下：

（1）打开 VS 2008→打开“服务器资源管理器”窗口→右击“数据连接”→选择“创建新 SQL Server 数据库”选项，如图 7.2 所示。

（2）在“创建新的 SQL Server 数据库”对话框中选择要链接的服务器名→输入新的数据库名→单击【确定】按钮，如图 7.3 所示，系统添加了一个数据库。

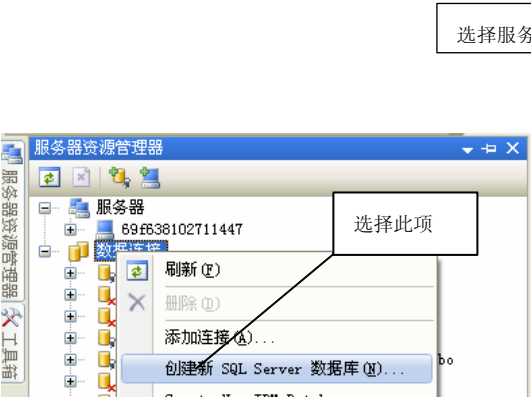


图 7.2 创建数据库步骤 1

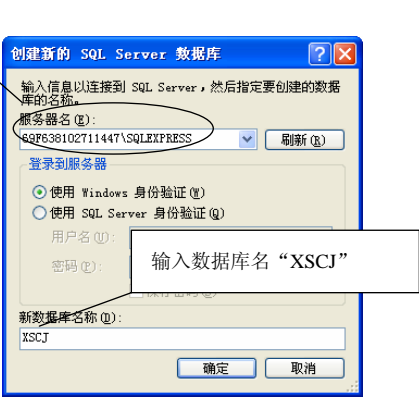


图 7.3 创建数据库步骤 2

注意：如果是本地服务器，则服务器名也可输入 localhost、127.0.0.1。

2. 创建表

用 VS 2008 创建表的步骤如下：

(1) 打开“服务器资源管理器”窗口→展开刚添加的数据库“XSCJ”→右击“表”→选择“添加新表”选项，如图 7.4 所示。

(2) 给新表设计字段和字段类型，设计后右击“XH”→选择“设置主键”选项，XH 被设置为新表的主键，如图 7.5 所示。其中“XH”、“XM”、“XB”、“ZY”、“ZXF”、“CSSJ”、“BZ”、“ZP”分别表示“学号”、“姓名”、“性别”、“专业”、“总学分”、“出生时间”、“备注”、“照片”。

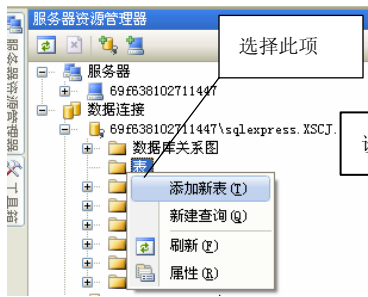


图 7.4 添加新表

列名	数据类型	允许 Null
XH	varchar (20)	<input type="checkbox"/>
XM	varchar (20)	<input checked="" type="checkbox"/>
XB	bit	<input checked="" type="checkbox"/>
ZY	varchar (100)	<input checked="" type="checkbox"/>
ZXF	int	<input checked="" type="checkbox"/>
CSSJ	datetime	<input checked="" type="checkbox"/>
BZ	varchar (500)	<input checked="" type="checkbox"/>
ZP	image	<input checked="" type="checkbox"/>

图 7.5 设计表的字段和字段类型

(3) 关闭窗口→保存对表格的修改，在弹出的“选择名称”对话框中输入“XS”，单击【确定】按钮，XS 表即创建成功。

以同样的方法创建 KC（课程）表，表的结构如图 7.6 所示。其中，“KCH”、“KCM”、“XQ”、“XS”、“XF”分别表示“课程号”、“课程名”、“学期”、“学时”、“学分”。

	列名	数据类型	允许 Null
	KCH	varchar (20)	<input type="checkbox"/>
	KCM	varchar (100)	<input checked="" type="checkbox"/>
	XQ	tinyint	<input checked="" type="checkbox"/>
	XS	int	<input checked="" type="checkbox"/>
	XF	int	<input checked="" type="checkbox"/>

图 7.6 KC 表的结构

注意：创建数据库时数据库服务应处于被打开状态。

7.1.4 SQL语言

结构化查询语言 SQL（Structured Query Language）是用于关系数据库操作的标准语言，虽然名为查询语言，但实际上具有数据定义、查询、更新和控制等多种功能。SQL 语言由 3 部分组成。

(1) 数据定义语言（Data Description Language, DDL）：用于执行数据库定义的任务，对数据库及数据库中的各种对象进行创建、删除、修改等操作。



(2) 数据操纵语言 (Data Manipulation Language, DML): 用于操纵数据库中的各种对象, 检索和修改数据。

(3) 数据控制语言 (DCL, Data Control Language): 用于安全管理, 确定哪些用户可以查看或修改数据库中的数据。

SQL 语句通常由一个描述要产生的动作的关键字开始, 如 Create、Select、Update 等。紧随语句的是一个或多个子句, 子句进一步指明语句对数据的作用条件、范围、方式等。

## 1. SELECT 查询

SELECT 查询是 SQL 语言的核心, 功能强大, 与各类 SQL 子句结合, 可完成各类复杂的查询操作。在数据库应用中, 最常用的操作是查询, 同时查询还是数据库的其他操作 (如统计、插入、删除及修改) 的基础。

### (1) SELECT 语句

SELECT 语句很复杂, 其主要的子句语法格式如下:

```
SELECT [DISTINCT] [别名.]字段名或表达式 [AS 列标题]
/* 指定要选择的列或行及其限定 */
FROM table_source
/* FROM 子句, 指定表或视图 */
[ WHERE search_condition ]
/* WHERE 子句, 指定查询条件 */
[ GROUP BY group_by_expression ]
/* GROUP BY 子句, 指定分组表达式 */
[ ORDER BY order_expression [ ASC | DESC ] ]
/* ORDER 子句, 指定排序表达式和顺序 */
```

其中, SELECT 和 FROM 子句是不可缺少的。

① SELECT 子句指出查询结果中显示的字段名, 以及字段名和函数组成的表达式等。可用 DISTINCT 去除重复的记录行; AS 列标题指定查询结果显示的列标题。若要显示表中所有字段, 则可用通配符 “\*” 代替字段名列表。

② WHERE 子句定义了查询条件。WHERE 子句必须紧跟在 FROM 子句之后。


从查询条件的构成可以看出, 可以将多个判定运算的结果通过逻辑运算符再组成更为复杂的查询条件。判定运算包括比较运算、模式匹配、范围比较、空值比较和子查询等。

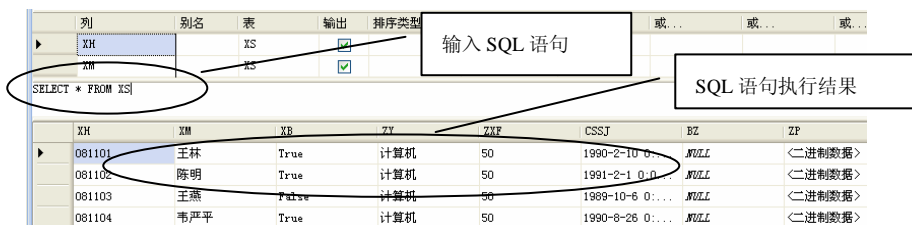
GROUP BY 子句和 ORDER BY 子句分别对查询结果分组和排序。

**【例 7-1】** 查询 XSCJ 数据库中的 XS 表的所有信息。

在 VS 2008 中查询, 步骤如下:

(1) 打开 VS 2008 → 打开 “服务器资源管理器” 窗口 → 右击刚添加的数据库 “XSCJ” → 选择 “新建查询” 选项, 在添加表对话框中选择 XS 表, 关闭对话框。

(2) 输入 SQL 语句, 单击 SQL 语句执行按钮 , 结果如图 7.7 所示。



列	别名	表	输出	排序类型	或...	或...	或...
XH		XS	<input checked="" type="checkbox"/>				
XM		XS	<input checked="" type="checkbox"/>				
SELECT * FROM XS							
XH	XM	XB	ZY	ZXF	CSSJ	BZ	ZP
081101	王林	True	计算机	50	1990-2-10 0...	NULL	<二进制数据>
081102	陈明	True	计算机	50	1991-2-1 0 0...	NULL	<二进制数据>
081103	王燕	False	计算机	50	1989-10-6 0...	NULL	<二进制数据>
081104	韦平平	True	计算机	50	1990-8-26 0...	NULL	<二进制数据>

图 7.7 SQL 语句执行结果

下面是常用的查询语句。

① 查询 Student 数据库的 XS 表中各个同学的姓名和总学分。

```
USE XSCJ
SELECT XM,ZXF FROM XS
```

② 查询表中所有记录。查询 XS 表中各个同学的所有信息。

```
SELECT * FROM XS
```

③ 条件查询。查询 XS 表中总学分大于等于 120 的同学的情况。

```
SELECT * FROM XS WHERE ZXF >=120
```

④ 多重条件查询。查询 XS 表中所在系为“计算机”且总学分大于等于 120 的同学的情况。

```
SELECT * FROM XS WHERE ZY ='计算机' AND ZXF>=120
```

⑤ 使用 LIKE 谓词进行模式匹配。查询 XS 表中姓“王”且单名的学生的情况。

```
SELECT * FROM XS WHERE XM LIKE '王_'
```

⑥ 用 BETWEEN...AND 指定查询范围。查询 XS 表中不在 1989 年出生的学生的情况。

```
SELECT * FROM XS WHERE CSSJ NOT BETWEEN '1989-1-1' and '1989-12-31'
```

⑦ 空值比较。查询总学分尚不定的学生的情况。

```
SELECT * FROM XS WHERE ZXF IS NULL
```

⑧ 查询结果分组。按专业计算总学分并按专业分组。

```
SELECT ZY ,sum(ZXF) FROM XS GROUP BY ZY
```

⑨ 查询结果排序。将计算机系的学生按出生时间先后排序。

```
SELECT * FROM XS WHERE ZY = '计算机' ORDER BY CSSJ
```

(2) 常用聚合函数

对表数据进行检索时，经常需要对结果进行汇总或计算，例如，在学生成绩数据库中求某门功课的总成绩，统计各分数段的人数等。聚合函数用于计算表中的数据，返回单个计算结果。常用的聚合函数列于表 7.2 中。

表 7.2 常用的聚合函数

函 数 名	说 明
AVG	求组中值的平均值
COUNT	求组中项数，返回 int 类型整数
MAX	求最大值

MIN	求最小值
SUM	返回表达式中所有值的和

**【例 7-2】** 使用聚合函数，对 XSCJ 数据库表执行查询。

① 求计算机系学生的平均学分。

```
SELECT AVG(ZXF) AS '平均学分' FROM XS WHERE ZY= '计算机'
```

② 求学生的最高学分和最低学分。

```
SELECT MAX(ZXF) AS '最高学分', MIN(ZXF) AS '最低学分' FROM XS
```

③ 求学生的总人数。

```
SELECT COUNT(*) AS '学生总数' FROM XS
```

2. 数据更新

数据更新语句包括 INSERT、UPDATE 和 DELETE 语句。数据更新语句一次只能对一个表进行更新，不能进行多表操作。

(1) 插入数据语句 INSERT

INSERT 可添加一个或多个记录至一个表中。INSERT 有两种语法格式。

● 语法格式 1:

```
INSERT INTO target [IN externaldatabase] (fields_list)
{DEFAULT VALUES|VALUES (DEFAULT | expression_list) }
```

● 语法格式 2:

```
INSERT INTO target [IN externaldatabase] fields_list
{SELECT...|EXECUTE...}
```

其中，target 为欲追加记录的表（Table）或视图（View）的名称；externaldatabase 为外部数据库的路径和名称；expression\_list 为需要插入的字段值表达式列表，其个数应与表的字段个数一致。若指定要插入值的字段 fields\_list，则应与 fields\_list 的字段个数相一致。

使用语法格式 1 将数据插入到表或视图的全面或者部分字段中。语法格式 2 的 INSERT 语句插入来自 SELECT 语句或来自使用 EXECUTE 语句执行的存储过程的结果集。

例如，以下语句向 XS 表添加一条记录：

```
INSERT INTO XS
(XH, XM, XB, ZY, ZXF, CSSJ, BZ, ZP)
VALUES ('081116', '罗亮', 1, '计算机', 150, '1/30/1989', '三好学生', NULL)
```

(2) 删除数据语句 DELETE

DELETE 语句用来从一个或多个表中删除记录。

DELETE 语句的语法格式如下：

```
DELETE FROM table_names [WHERE ...]
```

例如，以下语句从 XS 表中删除姓名为“罗亮”的记录：

```
DELETE FROM XS WHERE XM = '罗亮'
```

### （3）更新数据语句 UPDATE

UPDATE 语句用来更新表中的记录。

UPDATE 语句的语法格式如下：

```
UPDATE table_name
SET Field_1=expression_1[,Field_2=expression_2...]
[FROM table1_name|view1_name[,table2_name|view2_name...]]
[WHERE...]
```

其中，Field 为需要更新的字段；Expression 表示要更新字段的新值表达式。

例如，以下语句将计算机系的学生总分增加 10：

```
UPDATE XS
SET ZXF= ZXF +10
WHERE ZY = '计算机'
```

## 7.2 数据访问技术

越来越多的 Web 应用程序需要与数据源中的数据进行交互，数据源的多样性和复杂性，导致数据访问技术面临很大的难度，而 ASP.NET 3.5 在数据访问的简单化、智能化、多样化、高效、高性能等方面都做了较大的改进。

### 7.2.1 数据访问概述

ASP.NET 1.x 可以在运行时把整个数据集合乃至任意的数据源（如简单的数组、复杂的 Oracle 数据库）绑定到控件上，从而减少数据访问及显示所必须的代码，但是仍然有大量的重复性代码需要编写，尤其是一些常用的功能，如对数据进行分页、排序、编辑、选择等都需要手动编写代码，给开发人员带来了极大的不便。ASP.NET 2.0 在扩展数据绑定的基础上，引入新的数据抽象层，称为数据源控件。同时添加了一系列新数据绑定控件，如 GridView、DetailsView 和 FormView 等。这些控件主要用来在无状态的 Web 模型中，完成显示和更新网页中的数据所需要的基本任务。

ASP.NET 通过 2 种途径来实现数据访问：一是使用 ADO.NET（即 System.Data 命名空间）和 System.Xml 命名空间中的类来访问普通数据源和 XML 数据源；二是通过数据源控件和数据绑定控件来访问数据源，完成显示和更新数据所需的基础任务，这种方案无须编写任何代码。

ASP.NET 3.5 在 ASP.NET 2.0 的基础上又引入了 LinqDataSource 和 ListView 控件，使得数据的访问和显示更加简单。

### 7.2.2 数据源控件简介

ASP.NET 2.0 通过数据源控件引入了一个新的抽象层，这些控件抽象了底层数据提供程

序的使用，向页面上的数据绑定控件提供来自不同数据源的数据，并且提供排序、分页、缓存、更新、插入和删除数据等功能，这样开发人员无须编写代码就能够利用这些功能设计页面。另外，由于数据源控件都派生于 **Control** 类，所以可以像其他 **Web** 服务器控件那样在 **HTML** 中明确定义和控制，也可以通过编程定义和控制数据源控件。与普通控件不同的是，数据源控件只用来表示特定的后端数据存储，并没有外在的呈现形式，即在运行中是不可见的，因此需要与数据绑定控件配合使用。

**ASP.NET 3.5** 包含 6 种类型的数据源控件，这些数据源控件允许用户使用不同类型的数据源。图 7.8 描述了 **ASP.NET 3.5** 的数据访问框架。

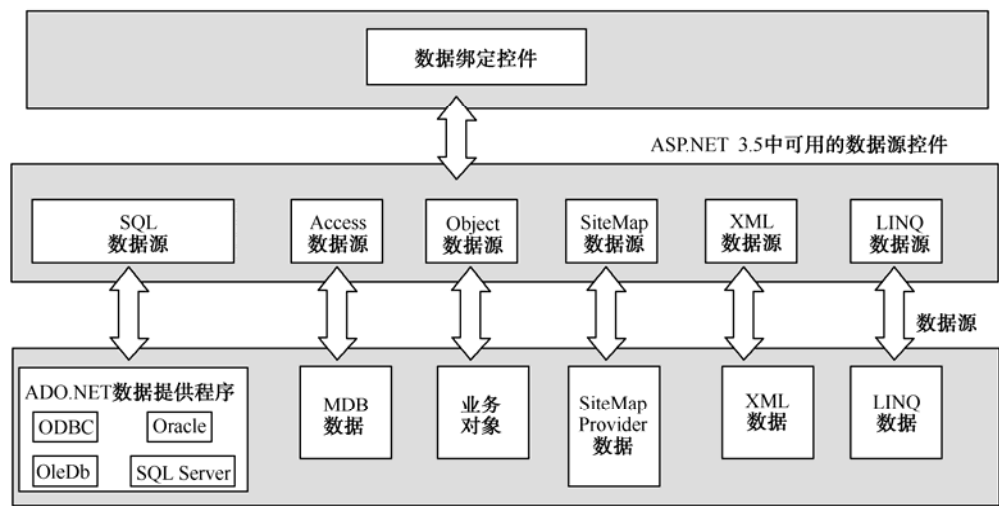


图 7.8 ASP.NET 3.5 的数据访问框架

### 7.2.3 数据绑定控件简介

要使用数据源控件，至少要有有一个数据绑定控件与它相绑定。数据绑定控件把数据源提供的数据作为标记，发送给请求的客户端浏览器，然后将数据呈现在浏览器页面上。数据绑定控件不仅能够自动绑定到数据源公开的数据，在页面请求生命周期的适当时间获取数据，而且有的控件还可以选择利用数据源提供的排序、分页、筛选、更新、删除和插入等功能。除了使用数据源控件作为数据绑定控件的数据源，传统的基于 **IEnumerable** 的数据容器，如 **DataView** 和集合等，都可以作为数据源与数据绑定控件一起使用。图 7.9 列出了数据绑定控件的层次结构。

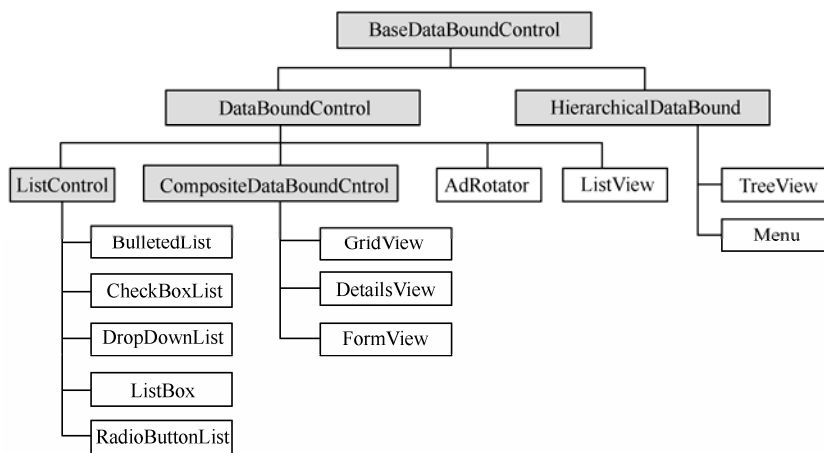


图 7.9 数据绑定控件的层次结构

## 7.3 数据源控件

数据源控件主要用来与数据源进行交互，通常数据源是数据库，也可以是数组、集合、XML 文件。数据源控件可以实现对不同类型数据源的数据访问，主要包括链接数据源、使用 SQL 语句获取和管理数据等。根据处理的数据源类型，数据源控件可分为 SqlDataSource、AccessDataSource、XmlDataSource、SiteMapDataSource、ObjectDataSource 和 LinqDataSource 控件。表 7.3 描述了 ASP.NET 3.5 中内置的 6 个数据源控件。

表 7.3 ASP.NET 3.5 中内置的数据源控件

数据源控件	说 明
SqlDataSource	允许访问支持 ADO.NET 数据提供程序的任意数据源，如 MS SQLServer、ODBC 或 Oracle。与 SqlServer 一起使用时支持高级缓存功能。当数据作为 DataSet 对象返回时，支持排序、筛选和分页
AccessDataSource	允许访问 Microsoft Access 数据库。当数据作为 DataSet 对象返回时，支持排序、筛选和分页
XmlDataSource	允许使用 XML 文件，特别适用于分层的 ASP.NET 服务器控件，如 TreeView 或 Menu 控件。支持使用 XPath 表达式的筛选功能，并允许对数据应用 XSLT 转换。可以通过保存更改后的整个 XML 文档来更新数据
SiteMapDataSource	可以对站点地图提供程序所存储的 Web 站点进行特定的站点地图数据访问
ObjectDataSource	支持绑定到中间层对象（如数据访问层或业务组件）来管理数据的 Web 应用程序，支持对其他数据源控件不可用的高级排序和分页方案
LinqDataSource	可以通过标记在 ASP.NET 网页中使用语言集成查询（LINQ），从数据对象中检索和修改数据。支持自动生成选择、更新、插入和删除命令，还支持排序、筛选和分页

### 7.3.1 SqlDataSource 控件

如果数据存储在 SQL Server、SQL Server Express、ODBC 数据源、OLE DB 数据源、Oracle 数据库中，就应该使用 SqlDataSource 控件。SqlDataSource 控件可以与其他数据绑定控件一起使用，开发人员用极少代码甚至不用代码，就可以在 ASP.NET 网页上显示和操作数据库。

SqlDataSource 数据源控件主要提供如下功能。

- 只需少量代码即可实现数据库操作：查询、插入、更新、删除。
- 以 DataReader 和 DataSet 方式返回查询结果集。
- 提供缓存功能。
- 提供冲突检测功能。

使用 SqlDataSource 的方法有：

- (1) 通过图形化的方式设置数据源控件的链接字符串和数据提供程序，以及所使用的 Sql 语句和参数；
- (2) 通过编程的方式设置数据源控件的链接字符串和数据提供程序，以及所使用的 Sql 语句和参数。

**【例 7-3】 图形化使用 SqlDataSource 控件。**

- (1) 运行 VS 2008，新建一个网站 “Chapter7”。
- (2) 添加一个命名为 “example7-3.aspx” 的网页，切换到设计视图。
- (3) 从工具箱的数据栏中拖放一个 SqlDataSource 控件、一个 GridView 控件和一个 Button 控件到页面上，右击 SqlDataSource 控件→选择 “配置数据源” 选项，在弹出的 “配置数据源” 对话框中单击【新建连接】按钮，出现 “添加连接” 对话框，在 “服务器名” 中选择所要连接的服务器，在 “连接到一个数据库” 中选择所要连接的数据库 “XSCJ”，单击【测试连接】按钮，测试连接是否成功，如图 7.10 所示。

注意：GridView 是一个以表格的形式自动显示记录集中数据的数据绑定控件。

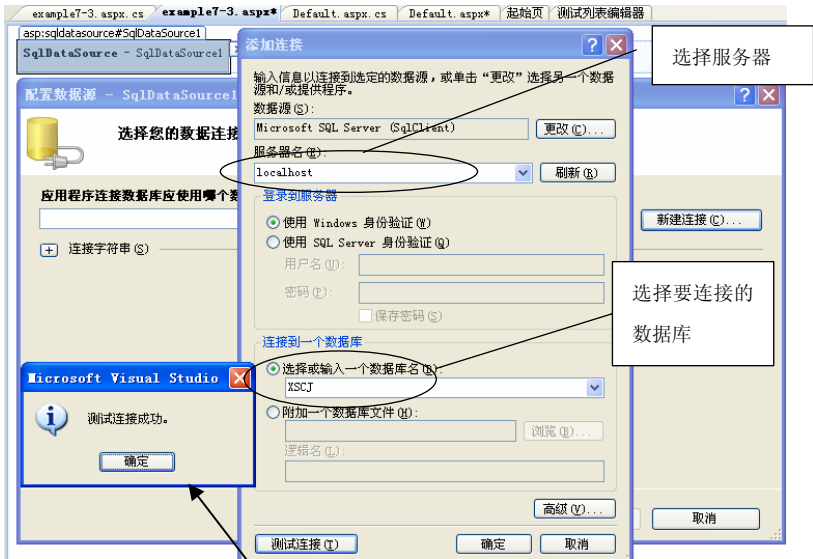


图 7.10 配置数据库连接

- (4) 确定后，单击【下一步】按钮，会提示是否把此连接保存到 Web.config 中，选择 “是” 以便以后使用；单击【下一步】按钮，出现 “配置 Select 语句” 界面，如图 7.11 所示。此界面提供 2 种设置 SQL 语句的选项：一种是 “指定自定义 SQL 语句或存储过程”，此选项可以为数据源控件自定义 SQL 语句或存储过程；另一种是 “指定来自表或视图的列”。选择第一种，会出现如

图 7.12 所示的界面，在 SELECT 选项卡中添加 SQL 语句 “select \* from kc where xf=@xf”。



图 7.11 “配置 Select 语句”界面

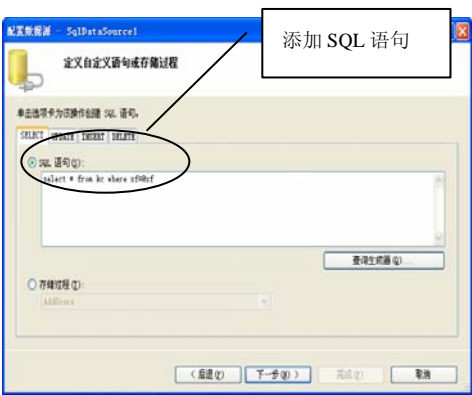


图 7.12 自定义 SQL 语句或存储过程

(5) 单击【下一步】按钮，如果 SQL 语句或存储过程中需要参数，则会出现“定义参数”界面，为参数指定参数值的来源，如图 7.13 所示，设置完毕后单击【下一步】按钮完成对数据源控件的设置。

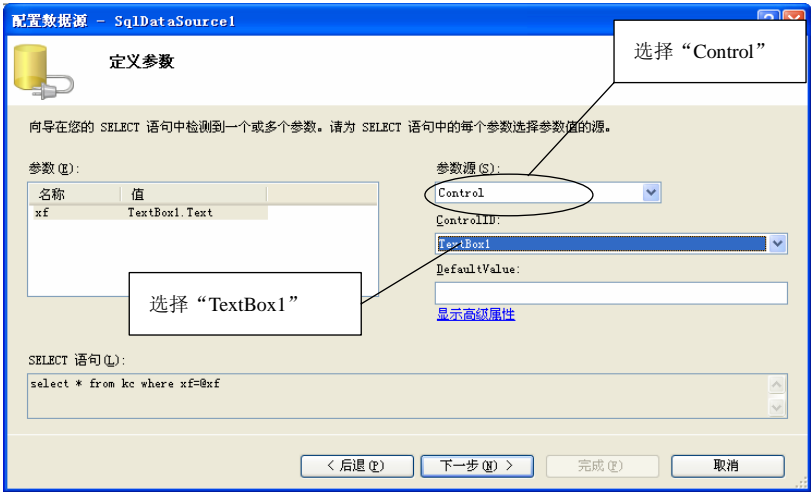


图 7.13 “定义参数”界面

SqlDataSource 参数源支持如下几种。

- Cookie: 把 Cookie 中的变量的值作为参数的值。
- Control: 把服务器控件的属性值作为参数的值。
- Form: 把 Form 表单内的元素的值作为参数的值。
- Profile: 把 Profile 文件中的属性值作为参数的值。
- QueryString: 把 QueryString 查询字符串中的变量值作为参数的值。
- Session: 把 Session 对象中的变量的值作为参数的值。

(6) 设置 GridView 的数据源为 SqlDataSource1，按【Ctrl+F5】组合键运行网页，在文本框内输入学分后单击 Button 控件，则 GridView 控件中就会显示对应的课程，结果如图 7.14 所示。打开 Web.config 文件，系统自动添加的数据库链接字符串如下所示：





图 7.14 运行结果

```
<connectionStrings>
<add name="XSCJConnectionString" connectionString="Data Source=localhost;Initial Catalog=
XSCJ;Integrated Security=True" providerName="System.Data.SqlClient" />
</connectionStrings>
```

**注意：**数据源控件绑定到数据绑定控件以后，数据绑定控件会自动调用数据源控件内的 SELECT 语句，无须手动调用。但是对于别的数据修改语句 UPDATE、DELETE、INSERT，则需要手动调用才会执行。

#### 【例 7-4】 编程使用 SqlDataSource 控件。

设计步骤如下：

- (1) 在网站“Chapter7”上添加 1 个命名为“example7-4.aspx”的网页，在页面内拖放一个 SqlDataSource 控件、一个 GridView 控件、一个 TextBox 控件和一个 Button 控件。
- (2) 打开代码页，在 Page\_Load()方法中添加如下代码：

```
//获取配置文件中的数据库链接字符串
SqlDataSource1.ConnectionString =
ConfigurationManager.ConnectionStrings["XSCJConnectionString"].ConnectionString;
SqlDataSource1.SelectCommand = "select * from KC where XF = @xf"; //设置操作 SQL 语句
SqlDataSource1.SelectParameters.Clear(); //清除参数，放置多次操作
SqlDataSource1.SelectParameters.Add("xf", TypeCode.String, TextBox1.Text); //给参数赋值
GridView1.DataSourceID = SqlDataSource1.ID; //把 GridView 绑定到 SqlDataSource 控件
```

- (3) 按【Ctrl+F5】组合键运行网页，在文本框内输入学分并单击 Button 控件，结果如图 7.14 所示。

### 7.3.2 AccessDataSource控件

AccessDataSource 控件是专门访问 Access 数据库的数据源控件，它与 SqlDataSource 控件类似，几乎不编写任何代码就可实现从 Access 数据库链接到显示、编辑数据等一系列功能。

AccessDataSource 控件的一个独特之处是不用设置 ConnectionString 属性。需要做的就是使用 DataFile 属性设置 Access 数据库的 mdb 文件位置，AccessDataSource 将负责维护数据库的基础链接。

### 7.3.3 XmlDataSource控件

SqlDataSource 和 AccessDataSource 控件主要是用来访问关系型数据库的,用这种方式组织的数据被称为“表格化数据”,其特点是扁平组织和存储。然而,还有另一种组织方式,被称为“层次化数据”,它是以树状模型来组织数据的,XML 文件就是这样的数据模型。

XmlDataSource 控件是专门用于访问 XML 文件的,它提供了绑定内存中或物理磁盘上的 XML 文档的一种简单方式。该控件有许多属性,便于指定包含数据的 XML 文件和用于把源 XML 转换为合适格式的 XSLT 转换文件;还可以提供一个 Xpath 查询,以选择某个数据子集。

**【例 7-5】** 使用 XmlDataSource 数据源控件访问和显示 XML 数据。

设计步骤如下:

(1) 在网站 Chapter7 中新建 1 个网页 example7-5.aspx,放入一个 XmlDataSource 控件和一个 DataList 控件。

(2) 打开“解决方案资源管理器”→右击“App\_Data”文件夹→选择“添加新项”→选择“XML 文件”→单击**【确定】**按钮。在创建的 XMLFile.xml 文件中添加代码(黑体部分):

```
<?xml version="1.0" encoding="utf-8" ?>
<channels>
  <channel >
    <title>百度</title>
    <link>http://www.baidu.com</link>
  </channel>
  <channel >
    <title>谷歌</title>
    <link>http://www.google.cn</link>
  </channel>
</channels>
```

(3) 为 XmlDataSource 配置数据源,设置 DataFile 属性为“~/App\_Data/XMLFile.xml”。DataList 的数据源选择“XmlDataSource1”。

(4) 切换到源视图,给 DataList 控件添加 ItemTemplate 模板代码:

```
<asp:DataList ID="DataList1" runat="server" DataSourceID="XmlDataSource1">
  <ItemTemplate>
    <a href='<%= XPath("link") %>' target="_blank"><b>
      <%=XPath("title")%></b></a>
  </ItemTemplate>
</asp:DataList>
```

(5) 按**【Ctrl+F5】**组合键运行网页,结果如图 7.15 所示。



图 7.15 访问 XML 数据

上面的代码中，在 ItemTemplate 模板中，数据绑定形式使用了 Xpath 表达式，例如，表达式 “XPath("link")” 绑定了 channel 节点中的 link 属性值。

### 7.3.4 SiteMapDataSource 控件

SiteMapDataSource 控件是用于检索站点地图文件的，它将获取的数据与站点导航控件结合，提供站点导航功能。与前面介绍的数据源控件不同，SiteMapDataSource 控件不需要设置数据源文件，也不需要设置查询条件。这主要是因为该控件只获取站点地图文件中的数据，而站点地图文件的名称和位置都是固定的，所以无须进行设置。有关 SiteMapDataSource 控件的应用见 6.3 节。

### 7.3.5 ObjectDataSource 控件

在实际的信息系统开发过程中，有些业务逻辑很复杂，不是简单的几条 SQL 语句就可以解决的，这时就需要采用功能灵活的 ObjectDataSource 数据源控件来构建多层的系统开发。

在复杂的多层系统中，数据库的操作通常被封装为一个业务逻辑类，处于业务逻辑层的位置，这个类中包括了对数据库的查询、插入、更新和删除等操作，已完全能够实现数据库的操作。但通过数据源控件能够和数据绑定控件很好地结合，无须代码就可以实现排序、分页等复杂功能，因此在此种情况下把业务逻辑类和 ObjectDataSource 进行管理，ObjectDataSource 不直接去操作数据库，而是通过调用业务逻辑类来实现数据的查询和更新操作。

**【例 7-6】** 利用 ObjectDataSource 控件操作数据库。

- (1) 在网站 Capture7 上添加一个页面，命名为 “example7-6.aspx”。
- (2) 打开“解决方案资源管理器”→选择“添加新项”→选择“类”，命名为“DataManager.cs”，单击【添加】按钮，系统提示是否把类存放到 App\_Code 文件夹，选择“是”。
- (3) 在 DataManager.cs 文件中添加命名空间 “using System.Data.SqlClient;”，添加代码如下所示（黑体部分）：

```
public class DataManager
{
    public DataManager()
    { // TODO: 在此处添加构造函数逻辑 }
    public DataSet getData(string value)
    {
```

```

        if (value != null)
        {
            SqlConnection conn = new SqlConnection();
            conn.ConnectionString = ConfigurationManager.
                ConnectionStrings["XSCJConnectionString"].ConnectionString;
            conn.Open();
            string sql = "select * from KC where XF = @xf";
            SqlCommand comm = new SqlCommand(sql, conn);
            comm.Parameters.AddWithValue("xf", value);
            DataSet dataset = new DataSet();
            SqlDataAdapter adapter = new SqlDataAdapter(comm);
            adapter.Fill(dataset);
            return dataset;
            comm.Dispose();
            conn.Close();
        }
        return null;
    }
}

```

(4) 在页面内拖放 1 个 ObjectDataSource 控件、1 个 GridView 控件、1 个 TextBox 控件和 1 个 Button 控件。

(5) 右击 ObjectDataSource→选择“配置数据源”，在弹出的“配置数据源”对话框中选择已经创建的“DataManager 类”，如图 7.16 所示，单击【下一步】按钮。



图 7.16 “选择业务对象”界面

(6) 在“定义数据方法”界面中指示为数据源所对应的操作指定业务对象内相应的方法，当调用数据源控件内对应的语句时，数据源控件会自动调用对应的方法，如图 7.17 所示，单击【下一步】按钮。



图 7.17 “定义数据方法”界面

(7) “定义参数”界面等同于 `SqlDataSource` 控件参数设置，如图 7.18 所示，单击【完成】按钮。



图 7.18 “定义参数”界面

(8) 选择 `GridView1` 的数据源为 `ObjectDataSource1`，按【Ctrl+F5】组合键运行网页，在文本框中输入“5”并单击 `Button` 按钮，结果同图 7.14。

注意：代码中涉及 ADO.NET 知识将在 7.5 节中为大家介绍。

### 7.3.6 LinqDataSource 控件

`LinqDataSource` 控件支持对数据对象的查询、添加、删除和更新等操作。`LinqDataSource` 控件的工作方式与其他数据源控件一样，也是把在控件上设置的属性转换为可以在目标数据对象上执行的查询。`SqlDataSource` 控件可以根据设置的属性生成 SQL 语句，`LinqDataSource` 控件也可以把设置的属性转换为有效的 LINQ 查询。

**【例 7-7】** 使用 `LinqDataSource` 控件查询数据。

(1) 在网站 `Capter7` 中新建网页 `example7-7.aspx`，放入一个 `LinqDataSource` 控件和一个 `GridView` 控件。

(2) 打开“解决方案资源管理器”→右击项目名→选择“添加新项”→选择“LINQ to SQL 类”→单击【确定】按钮。系统提示是否把“`DataClasses.dbml`”存放在“`App_Code`”文件夹中，选择“是”。在服务器资源管理器中选择 `XSCJ` 数据库中的 `XS` 表，将其拖放到对象关系设计器中并保存，如图 7.19 所示。

(3) 配置 LinqDataSource 的数据源。在图 7.20 中选择“DataClassesDataContext 上下文对象”，然后单击【下一步】按钮。在“配置数据选择”界面中选择 XS 表，选择字段，如图 7.21 所示，单击【完成】按钮。

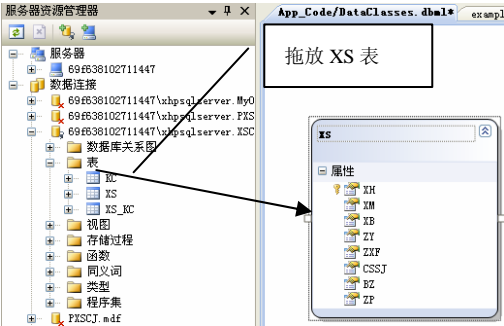


图 7.19 拖放 XS 表



图 7.20 “选择上下文对象”界面

(4) 为 GridView 控件的数据源属性指定 LinqDataSource 控件。按【Ctrl+F5】组合键运行网页，结果如图 7.22 所示。



图 7.21 配置数据选择对话框

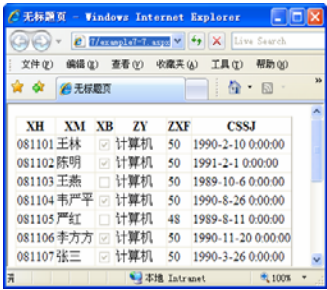


图 7.22 example7-7 网页运行的结果

## 7.4 数据绑定控件

ASP.NET 提供了多种数据绑定控件，用来显示数据。这些控件以丰富的表现形式将数据显示在页面中。

### 7.4.1 GridView 控件

GridView 控件以表格的形式显示数据，通过属性的设置，无须编程就能实现数据的分页、排序和编辑等功能。通过使用 GridView 控件，可以显示、编辑和删除多种不同数据源（如数据库、XML 文件和业务对象）中的数据。

可以使用 GridView 来完成以下操作：

- 通过数据源控件自动绑定和显示数据；
- 通过数据源控件对数据进行选择、排序、分页、编辑和删除。

另外，还可以通过以下方式自定义 GridView 控件的外观和行为：

- 指定自定义列和样式；
- 利用模板创建自定义用户界面 (UI) 元素；

● 通过处理事件将自己的代码添加到 GridView 控件的功能中。


【例 7-8】 使用 GridView 控件查询、删除数据。

操作步骤如下：

(1) 在网站 Capter7 中新建网页 example7-8.aspx，放入一个 SqlDataSource 控件和一个 GridView 控件。

(2) 右击 SqlDataSource 控件→选择“XSCJConnectionString”数据连接→单击【下一步】按钮，在“配置 Select 语句”界面中选择“指定自定义 SQL 语句或存储过程”单选框，在“SELECT”和“DELETE”选项卡中分别添加 SQL 语句，单击【下一步】按钮→单击【完成】按钮。SQL 如下所示：

```
SELECT [XH], [XM], [XB], [ZY], [ZXF] FROM [XS]      //SELECT 选项卡中添加
DELETE FROM [XS] WHERE [XH]=@XH                 //DELETE 选项卡中添加
```

(3)单击 GridView 控件右上角的图标，选择 GridView 控件的数据源为 SqlDataSource1，选择“启用分页”、“启用排序”、“启用删除”复选框，单击“编辑列”选项，如图 7.23 所示。

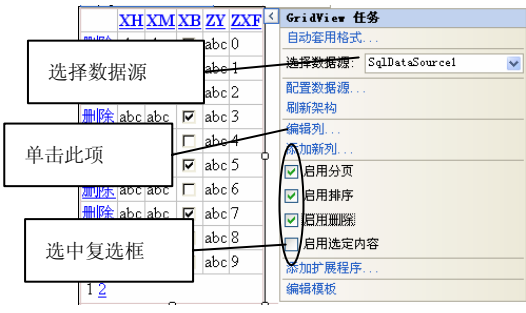


图 7.23 设置 GridView 任务

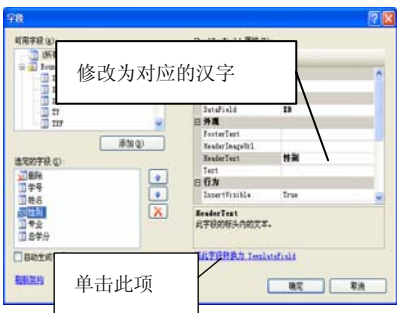


图 7.24 编辑字段

(4)在字段编辑对话框中将各个字段的 HeaderText 的属性设置为对应的汉字，并且把“性别”字段转换为“TemplateField”，如图 7.24 所示。

(5)右击 GridView 控件→选择“编辑模板”→选择“Column[3]-性别”，在“ItemTemplate”中的 CheckBox1 后输入“男”，如图 7.25 所示。右击性别编辑模板→选择“结束模板编辑”。

(6) 按【Ctrl+F5】组合键运行网页，结果如图 7.26 所示，可以单击各个字段排序；单击“删除”，即删除此行的学生。

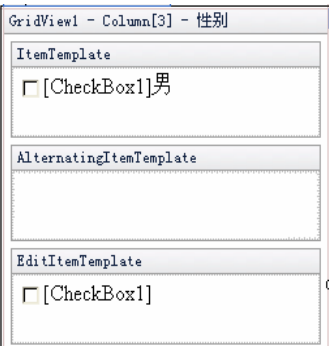


图 7.25 编辑性别模板

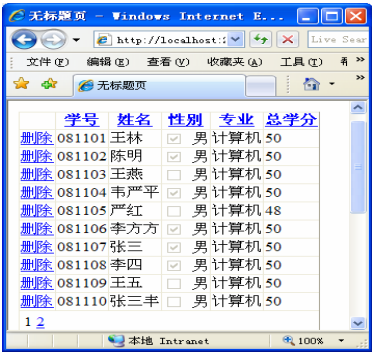


图 7.26 example7-8 网页的运行结果

## 7.4.2 DetailsView控件

**GridView** 控件有一个明显的特点，就是一次可以从数据源中获取大量数据并显示。这样做虽然方便，但显示的数据较多，会增加服务器的负载，用户也会觉得眼花缭乱。有时用户更希望一次只看到某一行数据的详细资料，而不是看到所有行的详细资料。**DetailsView** 控件可以显示一条记录的数据，可以利用 **DetailsView** 控件和 **GridView** 控件的配合使用来实现主详细信息显示的功能。

**DetailsView** 控件是以单条记录的方式来显示数据的，同时它也支持分页和编辑功能，但是其分页并非是因为数据行太多而需要分页，其实是分到下一条数据上。如果 **DetailsView** 控件是用来为 **GridView** 控件作详细数据补充显示的话，那么分页就没有用了，因为此时 **DetailsView** 控件数据源可能就只查询了一条记录。当然，如果一次查询的是多条记录，那么分页就有用了，可以将每一条记录分为一页。

**DetailsView** 控件一行显示一个字段，每个数据行是通过声明一个行字段控件创建的。不同的行字段类型确定了控件中各行的行为，共包含 7 种行字段类型。

(1) **BoundField**: 常用于以普通文本形式显示数据源中某个字段的值。

(2) **CheckBoxField**: 在 **DetailsView** 控件中显示一个复选框，通常用来显示布尔型数据字段。

(3) **CommandField**: 在 **DetailsView** 控件中用来显示执行编辑、插入、删除的内置命令按钮。

(4) **ImageField**: 在 **DetailsView** 控件中显示图片。

(5) **HyperLinkField**: 将数据源中某个字段的值显示为超链接。此行字段类型允许将另一个字段绑定到超链接的 URL。

(6) **ButtonField**: 在 **DetailsView** 控件中显示一个命令按钮，允许显示一个带有自定义按钮（如“添加”或“移除”按钮）控件的行。

(7) **TemplateField**: 根据指定的模板，为 **DetailsView** 控件中的行显示用户自定义的内容。此行字段类型用于创建自定义的行字段。

**注意**: 以上 7 个行字段只有当 **DetailsView** 控件的 **AutoGenerateRows** 属性设置为 **false** 时才能使用，该属性默认为 **Ture**，即允许 **DetailsView** 控件自动生成数据行。

**【例 7-9】** 使用 **GridView**、**DetailsView** 控件实现 **XS** 表的主从查询和修改 **BZ**（备注）内容。

操作步骤如下：

(1) 在网站 **Capter7** 中新建网页 **example7-9.aspx**，切换到设计视图，添加 1 行 2 列的表，在表的左边放入 1 个 **GridView** 控件，右边放入 1 个 **DetailsView** 控件，在表外还要添加 2 个 **SqlDataSource** 控件。

(2) 右击 **SqlDataSource1** 控件→选择“**XSCJConnectionString**”数据连接→单击【下一步】按钮，在“配置 **Select** 语句”界面中选择“指定来自表或者视图的列”单选框，选择“**XS**”表，选择“**XH**”、“**XM**”列。单击【下一步】按钮→单击【完成】按钮。

(3) 右击 **SqlDataSource2** 控件→选择“**XSCJConnectionString**”数据连接→单击【下一步】按钮，在“配置 **Select** 语句”界面中选择“指定自定义 **SQL** 语句或存储过程”单选框，在



“SELECT”和“UPDATE”选项卡中分别添加 SQL 语句，单击【下一步】按钮，在“定义参数”界面中设置参数源，如图 7.27 所示。单击【下一步】按钮→单击【完成】按钮。SQL 语句如下所示：

```
//SELECT 选项卡中添加
SELECT [XH], [XM], [XB], [ZY], [ZXF], [CSSJ], [BZ] FROM [XS] WHERE ([XH] = @XH)
//UPDATE 选项卡中添加
UPDATE [XS] SET [BZ]=@BZ WHERE ([XH]=@XH)
```

(4) 单击 GridView 控件右上角的图标，选择数据源为 SqlDataSource1，选择“启用分页”和“启用选定内容”复选框。单击 DetailsView 控件右上角的图标，选择数据源为 SqlDataSource2，选择“启用编辑”复选框。

(5) 打开 example7-9.aspx.cs 代码页，在 Page\_Load 方法中添加代码，如下所示：

```
if (!Page.IsPostBack) //如果第一次加载页面，则执行下面代码
{
    if (GridView1.Rows.Count > 0)
    {
        GridView1.SelectedIndex = 0; //选择第一个学生
    }
}
```

(6) 按【Ctrl+F5】组合键运行网页，单击【编辑】按钮，如图 7.28 所示，修改 BZ（备注）。单击【更新】按钮，可以修改学生备注。

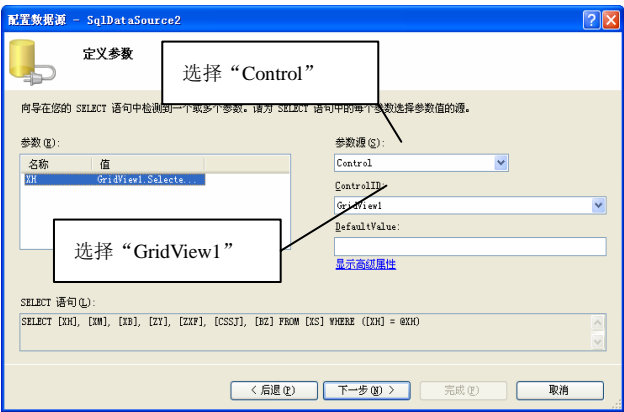


图 7.27 “定义参数”界面

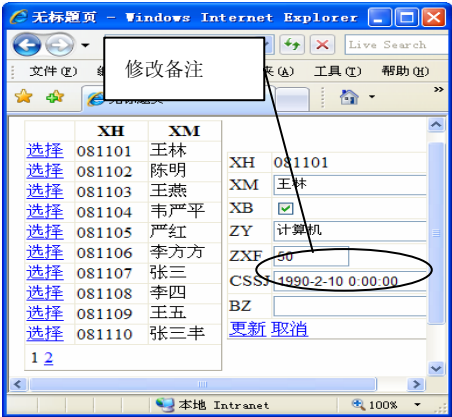


图 7.28 example7-9 网页运行结果

注意：数据库中的“XH”字段设置为关键字。

7.4.3 FormView控件

FormView 控件与 DetailsView 控件类似，也可以显示一条记录的数据，它们的不同点表现为 DetailsView 具有内置呈现机制，FormView 则需要自定义模板来定制字段显示和自定义命令按钮，可以更多地控制数据的显示和编辑方式。FormView 控件支持的模板如表 7.4 所示。

表 7.4 FormView 控件支持的模板

模板类型	说明
------	----

EditItemTemplate	编辑数据时的显示模板
EmptyDataTemplate	数据项为空时显示的模板
FooterTemplate	定义脚注行的内容
HeaderTemplate	定义标题行的内容
ItemTemplate	呈现只读数据时的模板
InsertItemTemplate	插入记录时的模板
PagerTemplate	启用分页功能时的模板

FormView 控件通常用于更新和插入记录，并且常常在主控件中列出详细信息的方案中使用。

**【例 7-10】** 使用 FormView 控件实现 XS 表查询和插入学生信息。



操作步骤如下：

(1) 在网站 Capter7 中新建网页 example7-10.aspx，切换到设计视图，添加一个 FormView 控件和一个 SqlDataSource 控件。

(2) 右击 SqlDataSource1 控件→选择“XSCJConnectionString”数据连接→单击【下一步】按钮，在“配置 Select 语句”界面中选择“指定自定义 SQL 语句或存储过程”单选框，在“SELECT”和“INSERT”选项卡中分别添加 SQL 语句，单击【下一步】按钮→单击【完成】按钮。SQL 语句如下所示：

```
SELECT [XH], [XM], [XB], [ZY] FROM [XS]
INSERT INTO [XS](XH,XM,XB,ZY)VALUES(@XH,@XM,@XB,@ZY)
```

//SELECT 选项卡中添加  
//INSERT 选项卡中添加

(3) 单击 FormView 控件右上角的图标，选择数据源为 SqlDataSource1，选择“启用分页”复选框。选择“编辑模板”，将字母改写为对应的汉字，单击右上角的图标，选择“InsertItemTemplate”，将字母改写为对应的汉字。结束模板编辑，如图 7.29 所示。（读者可以在模板中添加表，根据需要排列各个字段显示的位置。）

(4) 按【Ctrl+F5】组合键运行网页，单击【新建】按钮，输入新的学生信息，单击【插入】按钮即添加了新的学生信息，如图 7.30 所示。

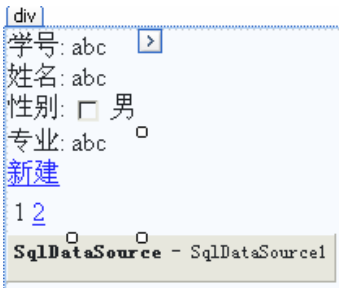


图 7.29 编辑后的 FormView 控件



图 7.30 添加新的学生信息

7.4.4 ListView控件

ListView 控件弥补了高度结构化的 GridView 控件和未结构化的 DataLis、Repeater 控件之间的空白。

利用 ListView 控件，可以绑定从数据源返回的数据项并显示它们。这些数据可以显示在

多个页面。可以逐个显示数据项，也可以将它们分组。

ListView 控件可以使用模板和样式来定义显示数据的格式。与 DataList 和 Repeater 控件相似，它也适用于任何具有重复结构的数据。但与这些控件不同的是，ListView 控件还允许用户编辑、插入和删除数据，以及对数据进行排序和分页，所有这一切都无须编写代码。ListView 控件共提供了 11 个模板，列于表 7.5 中。

表 7.5 ListView 控件的模板

模 板 名 称	说 明
LayoutTemplate	该模板用做 ListView 的根容器，用于控制数据项的整体布局
ItemTemplate	为控件中的每个数据项绑定内容
ItemSeparatorTemplate	用于提供在各个项之间的分隔符 UI
GroupTemplate	用于为组布局的内容提供 UI
GroupSeparatorTemplate	用于提供组之间的分隔符 UI
EmptyItemTemplate	在使用 GroupTemplate 模板时为空项的呈现提供 UI
EmptyDataTemplate	绑定的数据对象不包含数据项时显示的模板
SelectedItemTemplate	为选中的数据项提供 UI
AlternatingItemTemplate	为交替项提供独特的 UI
EditItemTemplate	为控件中编辑已有的项提供一个 UI
InsertItemTemplate	为控件中插入一个新数据项提供一个 UI

通过控件 DataPager，可以为 ListView 控件提供分页功能。DataPager 控件用于给终端用户显示分页的导航，并与实现了 IPagableItemContainer 接口的数据绑定控件（在 ASP.NET 3.5 中就是 ListView 控件）一起完成数据分页任务。事实上，如果在 ListView 控件的配置对话框中选择 Paging 复选框，则激活 ListView 控件上的分页功能，就会自动在 ListView 控件的 LayoutTemplate 模板中插入一个新的 DataPager 控件。


ListView 控件功能很强，使用它可以在不编写任何代码的情况下，对数据进行灵活的布局，同时实现对数据的排序、分页和编辑等操作。

**【例 7-11】** 使用 ListView 控件实现查询、插入、更新和删除 XS 表中的学生信息。

操作步骤如下：

（1）在网站 Capter7 中新建网页 example7-11.aspx，切换到设计视图，添加一个 ListView 控件和一个 SqlDataSource 控件。

（2）右击 SqlDataSource1 控件→选择“XSCJConnectionString”数据连接→单击【下一步】按钮，在“配置 Select 语句”界面中选择“指定来自表或视图的列”单选框，选择“XS”表，选择除“ZP”的所有的列，单击【高级】按钮→选择“生成 INSERT、UPDATE 和 DELETE “语句” →单击【确定】按钮→单击【下一步】按钮→单击【完成】按钮。

（3）单击 ListView 控件右上角的“”图标，选择数据源为 SqlDataSource1，选择“配置 ListView”选项，选择“启用编辑”、“启用插入”、“启用删除”和“启用分页”复选框，单击【确定】按钮，

（4）按【Ctrl+F5】组合键运行网页，结果如图 7.31 所示。

XH	XM	XB	ZY	ZXF	CSSJ	BZ
081101	王林	<input checked="" type="checkbox"/> 计算机	50	1990-2-10 0:00:00		
081102	陈明	<input checked="" type="checkbox"/> 计算机	50	1991-2-1 0:00:00		
081103	王燕	<input type="checkbox"/> 计算机	50	1989-10-6 0:00:00		
081104	张严平	<input checked="" type="checkbox"/> 计算机	50	1990-8-26 0:00:00		
081105	严红	<input type="checkbox"/> 计算机	48	1989-6-11 0:00:00		有一门功课不及格, 待补考。
081106	李方方	<input checked="" type="checkbox"/> 计算机	50	1990-11-20 0:00:00		
081107	张三	<input checked="" type="checkbox"/> 计算机	50	1990-3-26 0:00:00		
081108	李四	<input checked="" type="checkbox"/> 计算机	50	1982-6-28 0:00:00		
081109	王五	<input type="checkbox"/> 计算机	50	1990-3-26 0:00:00		
081110	张三丰	<input type="checkbox"/> 计算机	52	1990-3-26 0:00:00		三好学生

图 7.31 example7-11 网页运行的结果

### 7.4.5 内部数据绑定语法

在 ASP.NET 1.x 中，为模板中的控件绑定数据主要使用如下语法：

```
<%# Container.DataItem("name") %>
<%# Container.Eval("name") %>
<%# Container.Eval(Container.DataItem, "HireDate", "{0:mm dd yyyy}") %>
```

在 ASP.NET 3.5 中，对内部数据绑定的语法进行了简化，还增加了新绑定工具。

#### 1. DataBinder语法的变化

ASP.NET 有 3 种数据绑定的方式。首先，可以继续使用 Container.DataItem 语法来绑定数据。这样可以兼容以前编写的程序。但如果编写新的页面，则可以采用简单的绑定形式，直接使用 Eval 方法：

```
<%# Eval("name") %>
```

也可以使用 Eval 方法的格式化方法来格式化数据：

```
<%# Eval("HireDate", "{0:mm dd yyyy}") %>
```

除了这些变化外，ASP.NET 还引入了一种新形式的数据绑定，称为双向数据绑定，是通过 Bind 方法实现的。它与 Eval 方法的工作过程类似，其语法如下：

```
<%# Bind("name") %>
```

新方法应在新控件 GridView、DetailsView、FormView 中使用，这些控件能自动更新数据源。

在使用数据绑定语句时，<%# %>界定符之间的所有内容都作为表达式来处理。这是很重要的，因为在数据绑定时，它提供了一个额外的功能，例如，可以追加额外的数据：

```
<%# "姓名：" + Eval("name") %>
```

或者给方法传送计算出来的值，例如，给 DoSomeProcess 方法传送参数：

```
<%# DoSomeProcess(Bind("name")) %>
```

## 2. XML数据绑定

XML 在应用程序中越来越普遍，所以 ASP.NET 也有几种专门用于绑定 XML 的新方式，这就是 XPathBinder 类。这些新的数据绑定表达式可以处理 XML 的层次格式。另外，除了不同的方法名外，其绑定方法的工作方式与前面介绍的 Eval 和 Bind 方法完全相同。这些绑定方法应在 XmlDataSource 控件中使用。

使用 XPathBinder 类的第一种绑定格式如下所示：

```
<%# XPathBinder.Eval(Container.DataItem,"employees/employee/Name") %>
```

XPathBinder 没有像 Eval 方法那样指定字段名，而是绑定 XPath 查询的结果。与标准的 Eval 表达式一样，XPathBinder 类的 Eval 数据绑定方法也有一种缩写格式：

```
<%# XPath ("employees/employee/Name") %>
```

另外，与 Eval 方法一样，XPath 也可以把格式应用于数据：

```
<%# XPath ("employees/employee/HireDate","{0:mm dd yyyy}") %>
```

XPathBinder 类使用所提供的 XPath 查询返回一个节点。如果要从 XmlDataSource 控件中返回多个节点，则可以使用 XPathBinder 类的 Select 方法。该方法返回匹配 XPath 查询的一个节点列表。其语法如下：

```
<%# XPathBinder.Select(Container.DataItem,"employees/employee ") %>
```

或者使用缩写格式：

```
<%# XPathSelect("employees/employee ") %>
```

## 7.5 ADO.NET数据访问编程模型

早期的数据处理主要依赖基于连接的双层模型，当数据处理越来越多地使用多层模型时，就必须考虑非连接方式下的数据处理模型，以提高应用程序的可伸缩性。ADO.NET 正是这样一种能支持 N 层的数据访问应用程序模型。

### 7.5.1 ADO.NET数据访问模型简介

ASP.NET 使用 ADO.NET 数据模型。该模型从 ADO 发展而来，但它不只是对 ADO 的改进，而是采用了一种全新的技术，主要表现在以下几个方面：

- (1) ADO.NET 不是采用 ActiveX 技术，而是与 .NET 框架紧密结合的产物。
- (2) ADO.NET 包含对 XML 标准的完全支持，这对于跨平台交换数据具有重要的意义。
- (3) ADO.NET 既能在与数据源连接的环境下工作，又能在断开与数据源连接的环境下工作。

ADO.NET 提供了面向对象的数据库视图,并且在 ADO.NET 对象中封装了许多数据库属性和关系。最重要的是,ADO.NET 通过多种方式封装和隐藏了很多数据库访问的细节。可以完全不知道对象在与 ADO.NET 对象交互,也不用担心数据移动到另外一个数据库或者从另一个数据库获得数据的细节问题。如图 7.32 所示,为 ADO.NET 架构总览。

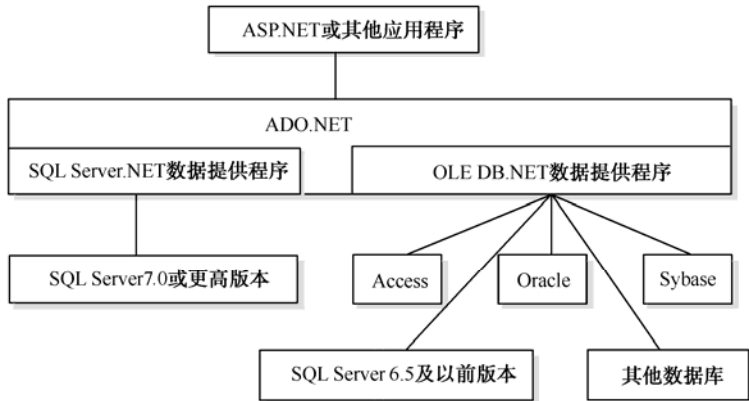


图 7.32 ADO.NET 架构总览

数据集是实现 ADO.NET 断开式连接的核心,从数据源读取的数据先缓存到数据集中,然后被程序或控件调用。数据源可以是数据库或者 XML 数据。

数据提供程序用于建立数据源与数据集之间的联系,它能连接各种类型的数据,并能按要求将数据源中的数据提供给数据集,或者从数据集向数据源返回编辑后的数据。

### 7.5.2 数据集

数据集相当于内存中暂存的数据库,不仅可以包括多张数据表,而且还可以包括数据表之间的关系和约束。允许将不同类型的数据表复制到同一个数据集中(其中某些数据表的数据类型可能需要做一些调整),甚至还允许将数据表与 XML 文档组合在一起协同操作。

数据集从数据源中获取数据以后就断开了与数据源之间的连接。允许在数据集中定义数据约束和表关系,增添、删除和编辑记录,还允许对数据集中的数据进行查询、统计等。当完成了各项数据操作以后,还可以将数据集中的最新数据更新到数据源。

数据集的这些特点为满足多层分布式应用的需要跨进了一大步。因为编辑和检索数据都是一些比较繁重的工作,所以需要跟踪列模式、存储关系数据模型等。如果在连接数据源的前提下完成这些工作,则不仅会使总体性能下降,而且还会影响到可扩展性的问题。

创建数据集对象的语句格式如下:

```
DataSet ds = new DataSet ();
```

或者

```
DataSet ds = new DataSet ("数据集名");
```

语句中 ds 代表数据集对象。可以通过调用 DataSet 的两个重载构造函数来创建 DataSet 的实例,并且可以选择指定一个名称参数。如果没有为 DataSet 指定名称,则该名称会设置为“NewDataSet”。

DataSet 对象最常用的属性是 Tables，通过该属性，可以获得或设置数据表行、列的值。  
例如：

ds.Tables["students"].Rows[i][j]

//表示访问 students 表的第 i 行第 j 列

DataSet 对象的常用方法有 Clear()和 Copy()，Clear()方法清除 DataSet 对象的数据，删除所有 DataTable 对象；Copy()方法复制 DataSet 对象的结构和数据，返回值是与本 DataSet 对象具有同样结构和数据的 DataSet 对象。  
在数据集中包括以下几种子类。

1. 数据表和数据表集合

(1) 数据表 (DataTable)

创建 DataTable 时，不需要为 TableName 属性提供值，可以在其他时间指定该属性，或者将其保留为空。但是，在将一个没有 TableName 值的表添加到 DataSet 中时，该表会得到一个从“Table”(表示 Table0)开始递增的默认名称 TableN。可以使用相应的 DataTable 构造函数创建 DataTable 对象。例如：

DataTable workTable = new DataTable("Customers"); //创建 DataTable 对象并指定名称为 Customers

表 7.6 列出了 DataTable 对象的常用属性和常用方法。

表 7.6 DataTable 对象的常用属性和常用方法

属性/方法	说 明
Columns	获取数据表的所有字段，即 DataColumnCollection 集合
DataSet	获取 DataTable 对象所属的 DataSet 对象
DefaultView	获取与数据表相关的 DataView 对象。DataView 对象可用来显示 DataTable 对象的部分数据。可通过对数据表选择、排序等操作获得 DataView（相当于数据库中的视图）
PrimaryKey	获取或设置数据表的主键
Rows	获取数据表的所有行，即 DataRowCollection 集合
TableName	获取或设置数据表名
Copy()	复制 DataTable 对象的结构和数据，返回与本 DataTable 对象具有同样结构和数据的 DataTable 对象
NewRow()	创建一个与当前数据表有相同字段结构的数据行
GetErrors()	获取包含错误的 DataRow 对象数组

(2) 数据表集合 (DataTableCollection)

DataSet 的所有数据表包含于数据表集合 DataTableCollection 中，通过 DataSet 的 Tables 属性访问 DataTableCollection。DataTableCollection 有以下两个属性。

- ① Count: DataSet 对象所包含的 DataTable 个数。
- ② Tables[index,name]: 获取 DataTableCollection 中下标为 index 或名称为 name 的数据表。例如：

<code>ds.Tables[0]</code>	//表示数据集对象 <code>ds</code> 中的第一个数据表
<code>ds.Tables[1]</code>	//表示数据集对象 <code>ds</code> 中的第二个数据表
<code>ds.Tables["students"]</code>	//表示数据集对象 <code>ds</code> 中名称为 “students” 的数据表

可以通过使用 `Add` 方法将 `DataTable` 添加到 `DataTable` 对象的 `Tables` 集合中，将其添加到 `DataSet` 中。例如：

<code>ds.Tables.Add("CustomersTable");</code>	//将数据表 <code>CustomersTable</code> 添加到数据集 <code>ds</code> 中
---	---

2. 数据行和数据行集合

(1) 数据行 (`DataRow`)

数据表中的每个数据行都是一个 `DataRow` 对象。`DataRow` 对象是给定数据表中的一行数据，或者说是数据表中的一条记录。`DataRow` 对象的方法提供了对表中数据的插入、删除、更新和查询等功能。提取数据表中的行的语句如下：

<code>DataRow dr = dt.Rows[n];</code>	//提取数据表中的行
---------------------------------------	------------

其中，`DataRow` 代表数据行类；`dr` 是数据行对象；`dt` 代表数据表对象；`n` 代表行的序号（序号从 0 开始）。

`DataRow` 对象的属性主要有：

- ① `Rows[index,columnName]`：获取或设置指定字段的值。
- ② `Table`：获取包含该数据行的 `DataTable` 对象。

`DataRow` 对象的方法主要有：

- ① `AcceptChanges()`：将所有变动过的数据行更新到 `DataRowCollection`。
- ② `Delete()`：删除数据行。
- ③ `IsNull({colName,index,Column 对象名})`：判断指定列或 `Column` 对象是否为空值。

(2) 数据行集合 (`DataRowCollection`)

数据表的所有行都被存放于数据行集合 `DataRowCollection` 中，通过 `DataTable` 的 `Rows` 属性访问 `DataRowCollection`。例如：

<code>stuTable.Rows[i][j]</code>	//表示访问 <code>stuTable</code> 表的第 <code>i</code> 行、第 <code>j</code> 列数据
----------------------------------	--

3. 数据列和数据列集合

(1) 数据列 (`DataColumn`)

数据表中的每个字段都是一个 `DataColumn` 对象。`DataColumn` 对象定义了表的数据结构。例如，可以用它确定列中的数据类型和大小，还可以对其他属性进行设置。例如，确定列中的数据是否是只读的、是否是主键、是否允许空值等；还可以让列在一个初始值的基础上自动增值，增值的步长也可以自行定义。

获取某列的值需要在数据行的基础上进行。其语句格式如下：

<code>string dc = dr.Columns["字段名"].ToString();</code>
--

或者



```
string dc = dr.Column[index].ToString();
```

两条语句具有同样的作用。其中 `dr` 代表引用的数据行，`dc` 是该行某列的值（用字符串表示），`index` 代表列（字段）对应的索引值（列的索引值从 0 开始）。

综合前面的语句，要取出数据表（`dt`）中第 3 行记录中的“姓名”字段，并将该字段的值放入一文本框（`textBox1`）中，语句可以写成：

```
DataTable dt = ds.Tables["Customers"]           //从数据集中提取数据表 Customers
DataRow dRow = dt.Rows[2];                       //从数据表中提取了第 3 行记录
string textBox1.Text=dRow["CompanyName"].ToString();//从行中取出名为 CompanyName 字段的值
```

表 7.7 列出了 `DataColumn` 对象的常用属性。

表 7.7 `DataColumn` 对象的常用属性

属 性	说 明
AllowDBNull	设置该字段可否为空值。默认值为 true
Caption	获取或设置字段标题。若未指定字段标题，则字段标题即为字段名
ColumnName	获取或设置字段名
DataType	获取或设置字段的数据类型
DefaultVale	获取或设置新增数据行时，字段的默认值.
ReadOnly	获取或设置新增数据行时，字段的值是否可修改。默认值为 false
Table	获取包含该字段的 <code>DataTable</code> 对象

通过 `DataColumn` 对象的 `DataType` 属性设置字段数据类型时，不可直接设置数据类型，而需按照以下语法格式：

```
DataColumn 对象名.DataType = typeof (数据类型)
```

其中的“数据类型”取值为 .NET Framework 数据类型。

(2) 数据列集合（`DataColumnCollection`）

数据表中的所有字段都被存放于数据列集合 `DataColumnColection` 中，通过 `DataTable` 的 `Columns` 属性访问 `DataColumnCollection`。例如：

```
stuTable.Columns[i].Caption           //代表 stuTable 数据表的第 i 个字段的标题
```

`DataColumnColection` 有以下 2 个属性：

- ① `Count`：数据表所包含的字段个数。
- ② `Columns[index,name]`：获取下标为 `index` 或名称为 `name` 的字段。例如：

```
DS.Tables[0].Columns[0]           //表示数据表 DS.Tables[0]中的第一个字段
DS.Tables[0].Columns["studentid"] //表示数据表 DS.Tables[0]的字段名为 studentid 的字段
```

`DataColumnColection` 的方法与 `DataTableCollection` 类似。

**【例 7-12】** 利用 `DataSet` 在内存中创建数据表 `Person`，给表定义 4 个字段：姓名、性别、专业、出生日期。向表内插入一条记录，并显示到页面上。

设计步骤如下：

- (1) 在网站 Chapter7 中加入一个页面“example7-12.aspx”，拖放一个 `GridView` 控件到页

面上。

(2) 在 Page\_Load 方法内添加如下代码：

```

        DataTable PSTab = new DataTable("Person");           //创建数据表 PSTab
        DataColumn PSCol = null;                             //声明数据列对象 PSCol
        string[] arrColName = { "姓名", "性别", "专业", "出生日期" };
        for (int i = 0; i < arrColName.Length; i++)           //设置表 Person 字段名
        {
            PSCol = new DataColumn();
            PSCol.ColumnName = arrColName[i];
            if (i != 3)                                         //如果不是出生日期则字段类型为 string
            {
                PSCol.DataType = typeof(System.String);      }
            else
            {
                PSCol.DataType = typeof(System.DateTime);    }
            PSTab.Columns.Add(PSCol);                           //将数据列 PSCol 添加到数据表 PSTab 中
        }
        DataRow row = PSTab.NewRow();                         //在 PSTab 中新建一行
        row["姓名"] = "王五";                                  //给此行“姓名”字段赋值
        row["性别"] = "男";                                    //给此行“男”字段赋值
        row[2] = "计算机";                                     //给此行第三个字段赋值
        row[3] = "1990-01-10";                                 //给此行第四个字段赋值
        PSTab.Rows.Add(row);                                   //将数据行 row1 添加到数据表 PSTab 中
        GridView1.DataSource = PSTab;                         //将数据表 PSTab 作为 GridView1 的数据源
        GridView1.DataBind();                                  //绑定到 GridView1 上
    
```

(3) 按【Ctrl+F5】组合键运行网页，结果如图 7.33 所示。

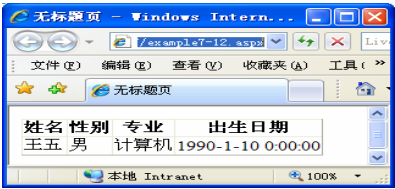


图 7.33 example7-12 网页的运行结果

7.5.3 数据提供程序

.NET Framework 数据提供程序用于链接到数据库、执行命令和检索结果，可以使用它直接处理检索到的结果，或将其放入 ADO.NET DataSet 对象。.NET Framework 数据提供程序是轻量的，它在数据源和代码之间创建一个最小层，在不牺牲功能为代价的前提下提高性能。表 7.8 列出了 .NET Framework 中包含的 .NET Framework 数据提供程序。

表 7.8 .NET Framework 数据提供程序

数据提供程序	说 明
SQL Server 数据提供程序	提供对 Microsoft SQL Server 7.0 版本或更高版本的数据访问。 System.Data.SqlClient 命名空间

OLE DB 数据提供程序	适合于使用 OLE DB 公开的数据源。System.Data.OleDb 命名空间
ODBC 数据提供程序	适合于使用 ODBC 公开的数据源。System.Data.Odbc 命名空间
Oracle 数据提供程序	适用于 Oracle 数据源。Oracle 数据提供程序支持 Oracle 客户端软件 8.1.7 版和更高版本。System.Data.OracleClient 命名空间

.NET Framework 数据提供程序包含 4 种核心对象，其名称及作用如下所述。

### 1. Connection

Connection 建立与数据库的连接。在进行数据库操作之前，首先要建立对数据库的连接。Connection 类中最重要的属性是 **ConnectionString**，该属性用来指定建立数据库连接所需要的连接字符串。ConnectionString 的主要参数如表 7.9 所示。

表 7.9 ConnectionString 的主要参数

参 数	说 明
Data Source	设置需连接的数据库服务器名
Initial Catalog	设置连接的数据库名称
Integrated Security	服务器的安全性设置，是否使用信任连接。值有 True、False 和 SSPI 三种，True 和 SSPI 都表示使用信任连接
Workstation Id	数据库客户端标识。默认为客户端计算机名

续表

参 数	说 明
Packet Size	获取与 SQL Server 通信的网络数据包的大小，单位为字节，有效值为 512~32 767，默认值为 8192
User ID	登录 SQL Server 的账号
Password(Pwd)	登录 SQL Server 的密码
Connection Timeout	设置 SqlConnection 对象连接 SQL 数据库服务器的超时时间，单位为秒。若在所设置的时间内无法连接数据库，则返回失败。默认为 15 秒

以 SQL Server 数据库的连接对象为例，类名为 SqlConnection，其创建的语句是：

SqlConnection conn = new SqlConnection();
 //创建 SQL Server 的连接对象 conn

设置 ConnectionString 属性的语句是：

conn.ConnectionString =  
 "Data Source=MySQLServer; //服务器名  
 user id=sa;password=123456; //安全信息  
 Initial catalog=XSCJ; Integrated Security=False"; //数据库名以其他参数

如上例，存储连接字符串的详细信息（如用户名和密码）可能会影响应用程序的安全性。若要控制对数据库的访问，一种较为安全的方法是使用 Windows 集成安全性，此时连接字符串可以修改为：

```
conn.ConnectionString =
"Data Source=MySQLServer;           //服务器名
Initial catalog=XSCJ;               //数据库名
Integrated Security=SSPI";          //采用 Windows 集成安全性
```

表 7.10 列出了 Connection 对象的常用方法。

表 7.10 Connection 对象的常用方法

方 法	说 明
Open()	打开与数据库的连接
Close()	关闭数据库连接
ChangeDatabase()	在打开连接的状态下，更改当前数据库
Dispose()	调用 Close()方法关闭与数据库的连接，并释放所占用的系统资源

注意：在完成连接后及时关闭连接，因为打开的连接占用了宝贵的系统资源。

2. Command

Command 是对数据源操作命令的封装。对于数据库来说，这些命令既可以是内联的 SQL 语句，也可以是数据库的存储过程。由 Command 生成的对象建立在连接的基础上，对连接的数据源指定相应的操作。

每个 .NET Framework 数据提供程序包括一个 Command 对象：OLEDB .NET Framework 数据提供程序包括一个 OleDbCommand 对象；SQL Server .NET Framework 数据提供程序包括一个 SqlCommand 对象；ODBC .NET Framework 数据提供程序包括一个 OdbcCommand 对象。Oracle .NET Framework 数据提供程序包括一个 OracleCommand 对象。

以下代码演示如何创建 SqlCommand 对象，以便从 SQL Server 中的 XSCJ 数据库的 XS 表中查找出所有学生的信息。

```
string sql = "SELECT * FROM XS";
SqlCommand command1 = new SqlCommand(sql, sqlConnection1);
```

参数 sql 为需执行的 SQL 命令。上述语句将生成一个命令对象 command1，对由 sqlConnection1 连接的数据源指定检索（SELECT）操作。这两个参数在创建 Command 对象时也可以省略不写，而在创建了 Command 对象后，再通过设置 Command 对象的 CommandText 和 CommandType 等属性来指定。Command 对象的常用属性和方法见表 7.11。

表 7.11 Command 对象的常用属性和方法

属性/方法	说 明
CommandText	取得或设置要对数据源执行的 SQL 命令、存储过程或数据表名
CommandType	获取或设置命令类别，可取值有 StoredProcedure、TableDirect、Text，代表的含义分别为存储过程、数据表名和 SQL 语句，默认为 Text
Connection	获取或设置 Command 对象所使用的数据连接属性
Parameters	SQL 命令参数集合

Cancel()	取消 Comand 对象的执行
CreateParameter	创建 Parameter 对象
ExecuteNonQuery()	执行 CommandText 属性指定的内容，返回数据表受影响行数
ExecuteReader()	执行 CommandText 属性指定的内容，返回 DataReader 对象
ExecuteScalar()	执行 CommandText 属性指定的内容，返回结果表第一行、第一列的值
ExecuteXmlReader()	执行 CommandText 属性指定的内容，返回 XmlReader 对象。只有 SQL Server 才能用此方法

Command 对象的 CommandType 属性用于设置命令的类别：可以是存储过程、表名或 SQL 语句。当将该属性值设为 CommandType.TableDirect 时，要求 CommandText 的值必须是表名而不能是 SQL 语句。例如：

```
OleDbCommand cmd = new OleDbCommand();
cmd.CommandText = "students";
cmd.CommandType = CommandType.TableDirect;
cmd.Connection = conn;
```

这段代码执行以后，将返回 students 表中的所有记录。它等价于以下代码：

```
OleDbCommand cmd = new OleDbCommand();
cmd.CommandText = "Select * from students";
cmd.CommandType = CommandType.Text;
cmd.Connection = conn;
```

3. DataReader

使用 DataReader 可以实现对特定数据源中的数据进行高速、只读、只向前的数据访问。与数据集（DataSet）不同，DataReader 是一个依赖于连接的对象。也就是说，它只能在与数据源保持连接的状态下工作。所有 DataReader 对象的基类均为 DbDataReader 类。

与 Command 类似，每个 .NET Framework 数据提供程序包括一个 DataReader 对象：OLE DB .NET Framework 数据提供程序包括一个 OleDbDataReader 对象；SQL Server .NET Framework 数据提供程序包括一个 SqlDataReader 对象；ODBC .NET Framework 数据提供程序包括一个 OdbcDataReader 对象；Oracle .NET Framework 数据提供程序包括一个 OracleDataReader 对象。

使用 DataReader 检索数据首先必须创建 Command 对象的实例，然后通过调用 Command 的 ExecuteReader 方法创建一个 DataReader，以便从数据源检索行。

以下示例说明如何使用 SqlDataReader，其中 command 代表有效的 SqlCommand 对象：

```
SqlDataReader reader = command.ExecuteReader();
```

在创建了 DataReader 对象后，就可以使用 Read 方法从查询结果中获取行。通过传递列的名称或序号引用，可以访问返回行的每一列。为了实现最佳性能，DataReader 也提供了一系列方法，使得能够访问其本机数据类型（GetDateTime、GetDouble、GetGuid、GetInt32 等）的列值。

以下代码示例循环访问一个 DataReader 对象，并从每个行中返回两个列：

```
if (reader.HasRows)                //判断是否有结果返回
while (reader.Read())              //依次读取行
{
    reader.GetInt32(0);
    reader.GetString(1);
}
reader.Close();
```

每次使用完 `DataReader` 对象后都应调用 `Close` 方法显式关闭。  
`DataReader` 对象的常用属性和方法见表 7.12。

表 7.12 `DataReader` 对象的常用属性和方法

属性/方法	说 明
IsClosed	获取 <code>DataReader</code> 对象的状态，为 <code>True</code> 表示关闭
Item({name,col})	获取或设置表字段值， <code>name</code> 为字段名， <code>col</code> 为列序号，序号从 0 开始
Close()	关闭 <code>DataReader</code> 对象
GetBoolean(Col)	获取序号为 <code>Col</code> 的列的值，所获取列的数据类型必须为 <code>Boolean</code> 类型；其他类似的方法还有 <code>GetByte</code> 、 <code>GetChar</code> 、 <code>GetDateTime</code> 、 <code>GetDecimal</code> 、 <code>GetDouble</code> 、 <code>GetFloat</code> 、 <code>GetInt16</code> 、 <code>GetInt32</code> 、 <code>GetInt64</code> 、 <code>GetString</code> 等
GetName(Col)	获取序号为 <code>Col</code> 的列的字段名
GetOrdinal(Name)	获取字段名为 <code>Name</code> 的列的序号
GetValue(Col)	获取序号为 <code>Col</code> 的列的值

续表

属性/方法	说 明
GetValues(values)	获取所有字段的值，并将字段值存放在 <code>values</code> 数组中
IsDBNull(Col)	若序号为 <code>Col</code> 的列为空值，则返回 <code>True</code> ，否则返回 <code>False</code>
Read()	读取下一条记录，返回布尔值。返回 <code>True</code> 表示有下一条记录，返回 <code>False</code> 表示没有下一条记录

4. `DataAdapter`

数据适配器（`DataAdapter`）利用连接对象（`Connection`）连接数据源，使用命令对象（`Command`）规定的操作从数据源中检索出数据送往数据集，或者将数据集中经过编辑后的数据送回数据源。所有 `DataAdapter` 对象的基类均为 `DbDataAdapter` 类。

如果所连接的是 `SQL Server` 数据库，则可以将 `SqlDataAdapter` 与关联的 `SqlCommand` 和 `SqlConnection` 对象一起使用，从而提高总体性能。对于支持 `OleDb` 的数据源，可以使用 `OleDbDataAdapter` 及其关联的 `OleDbCommand` 和 `OleDbConnection` 对象。对于支持 `ODBC` 的数据源，可使用 `OdbcDataAdapter` 及其关联的 `OdbcCommand` 和 `OdbcConnection` 对象。对于 `Oracle` 数据库，可使用 `OracleDataAdapter` 及其关联的 `OracleCommand` 和 `OracleConnection` 对象。

定义 `DataAdapter` 对象的语法格式有 4 种：

```
OleDbDataAdapter 对象名 = new OleDbDataAdapter()
OleDbDataAdapter 对象名 = new OleDbDataAdapter(OleDbCommand 对象)
OleDbDataAdapter 对象名 = new OleDbDataAdapter(SQL 命令, OleDbConnection 对象)
```

创建 SqlDataAdapter 对象语法格式与之类似，只要将所有的“OleDb”改为“Sql”即可。DataAdapter 有一个重要的 Fill 方法，此方法将数据填入数据集，语句如下：

```
dataAdapter1.Fill (dataSet1, "Products")
```

其中，dataAdapter1 代表数据适配器名；dataSet1 代表数据集名；Products 代表数据表名。当 dataAdapter1 调用 Fill()方法时将使用与之相关联的命令组件所指定的 SELECT 语句从数据源中检索行，然后将行中的数据添加到 DataSet 中的 DataTable 对象中。如果 DataTable 对象不存在，则自动创建该对象。

如果调用 Fill()方法之前与数据库的连接已经关闭，则将自动打开它以检索数据，不需要用语句将连接对象打开，执行完毕后再自动将其关闭。如果调用 Fill()方法之前连接对象已经打开，则检索后继续保持打开状态。

注意：一个数据集中可以放置多张数据表。但是每个数据适配器只能够对应于一张数据表。

**【例 7-13】** 下面是一个简单的 ADO.NET 应用程序，实现从数据源中检索数据并显示到网页中。

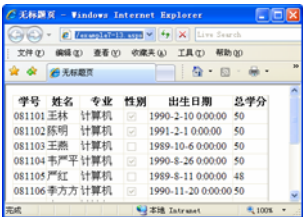
- (1) 在网站 Chapter7 中添加一个网页“example7-13.aspx”，拖放 1 个 GridView 控件到此页面上。
- (2) 在“example7-13.aspx.cs”代码页中添加如下命名空间：

```
using System.Data.SqlClient;
```

- (3) 在 Page\_Load 方法中添加如下代码：

```
SqlConnection conn = new SqlConnection(); //创建连接对象
//设置数据库连接字符串
conn.ConnectionString = "Data Source=(local);Initial Catalog=XSCJ;Integrated Security=SSPI";
conn.Open(); //打开连接
string sql = " SELECT XH as 学号, XM as 姓名, ZY as 专业, XB as 性别, CSSJ as 出生日期, ZXF as 总学分 FROM XS"; //定义 SQL 语句
SqlCommand comm = new SqlCommand(sql, conn); //定义命令对象
SqlDataReader reader = comm.ExecuteReader(); //执行命令对象
//把返回的结果集通过数据绑定控件显示出来
GridView1.DataSource = reader;
GridView1.DataBind();
conn.Close(); //关闭连接
```

- (4) 按【Ctrl+F5】组合键运行网页，运行结果如图 7.34 所示。



## 习 题

1. ASP.NET 3.5 提供了哪几个数据源控件？它们的主要用途是什么？
2. ASP.NET 3.5 提供了哪些数据绑定控件？它们具有什么功能？
3. 在 ASP.NET 3.5 中，对内部数据绑定的语法进行了哪些简化？
4. ADO.NET 数据访问模型提供了哪两个核心组件？它们的作用是什么？
5. DataReader 对象可以从数据库中读取由 SELECT 命令返回的只读、\_\_\_\_\_的数据集。
6. Command 对象使用 SELECT、INSERT、\_\_\_\_、DELETE 等数据命令与数据源通信。
7. \_\_\_\_\_对象充当数据库和 ADO.NET 对象模型中非连接对象之间的桥梁，能够用来保存和检索数据。
8. 要访问 oracle 数据库，需要使用（ ）命名空间。
  - A. System.Data.Oracle
  - B. System.Data.OracleClient
  - C. System.Data.ODBC
  - D. System.Data.SqlClient
9. 通过 SqlCommand 调用存储过程时，需要将其（ ）属性设置为 CommandType. Stored Procedure。
  - A. CommandType
  - B. CommandText
  - C. connection
  - D. SqlParameterCollection
10. 一个 DataSet 可以有（ ）DataTable。
  - A. 1 个
  - B. 2 个
  - C. 多个
  - D. 都不对
11. XmlDataSource 的\_\_\_\_\_属性用于指定 XML 文件来加载 XML 数据。
12. 下面的（ ）对象可以赋值给 DataSource 属性，进行数据绑定。
  - A. ArrayList 对象
  - B. DataReader 对象
  - C. DataRow 对象
  - D. DataTable 对象
13. 试述利用 ADO.NET 访问数据源的基本步骤。



# 第 8 章 文件I/O与流处理

在许多 Web 应用中都需要处理本地文件系统、读取文件夹结构、读写文件，或者进行其他与文件相关的操作。本章将介绍如何使用 `System.IO` 名称空间的类来管理本地文件系统上的文件和文件夹，如何使用各种 `Stream` 类进行文件的读写操作，以及如何进行文件的上传和邮件的发送。

## 8.1 文件系统操作

.NET Framework 中的 `System.IO` 名称空间用于处理与文件 I/O 相关的功能，可以灵活方便地管理文件系统上的驱动器、文件夹和文件。表 8.1 中列出了对文件系统进行操作的相关类。使用这些类可以浏览驱动器、文件夹和文件列表，进行文件夹与文件的创建、复制、删除等操作。

表 8.1 `System.IO` 名称空间的文件系统操作相关类

类 名	说 明
<code>DriveInfo</code>	提供对有关驱动器的信息访问的实例方法
<code>DirectoryInfo</code>	提供用于创建、移动和枚举文件夹和子文件夹的实例方法
<code>Directory</code>	提供创建、移动和枚举文件夹和子文件夹的静态方法
<code>FileInfo</code>	提供创建、复制、删除、移动和打开文件的实例方法，帮助创建 <code>FileStream</code> 对象
<code>File</code>	提供创建、复制、删除、移动和打开文件的静态方法，协助创建 <code>FileStream</code> 对象

在使用 `System.IO` 名称空间中的类时需要注意，ASP.NET 应用程序是在服务器上执行的，因此所访问的文件系统是运行 Web 应用程序的服务器端的文件系统，无法访问终端用户的文件系统。

### 8.1.1 使用驱动器

.NET 2.0 以上的类库中新增了 `DriveInfo` 类，该类增强了 .NET Framework 以前版本中 `Directory` 类的 `GetLogicalDrives()` 方法，使用它可以获得服务器的本地文件系统注册的信息，如每个驱动器的名称、类型、容量和状态等信息。`DriveInfo` 类的主要成员见表 8.2。

表 8.2 `DriveInfo` 类的主要成员

成 员 名	说 明
<code>AvailableFreeSpace</code>	指示驱动器上的可用空闲空间量，以字节为单位
<code>DriveFormat</code>	获取文件系统的名称，如 NTFS 或 FAT32
<code>DriveType</code>	获取驱动器类型，如固定硬盘、CD-ROM、可移动硬盘或者未知类型
<code>IsReady</code>	指示驱动器是否已准备好
<code>Name</code>	获取分配给驱动器的名称，如 C:\或 E:\

成 员 名	说 明
RootDirectory	获取驱动器的根文件夹
TotalFreeSpace	获取驱动器上的可用空闲空间总量，而不只是当前用户可用的空闲空间量
TotalSize	获取驱动器上存储空间的总大小
VolumeLabel	获取或设置驱动器的卷标
GetDrives()	检索计算机上的所有逻辑驱动器的驱动器名称

【例 8-1】 使用 DriveInfo 类显示本地驱动器的信息。

设计步骤如下：

(1) 运行 VS 2008，新建一个网站 “Chapter8”。

(2) 添加一个命名为 “example8-1.aspx” 的网页，打开 example8-1.aspx.cs 代码页，添加命名空间：

```
using System.IO;
```

(3) 在 Page\_Load 方法体内添加如下代码：

```
DriveInfo drive = new DriveInfo("D");
Response.Write("驱动器名: "+drive.Name);
Response.Write("<br>驱动器类型: " + drive.DriveType.ToString());
Response.Write("<br>驱动器可用空间: " + drive.AvailableFreeSpace.ToString());
Response.Write("<br>驱动器格式: " + drive.DriveFormat);
Response.Write("<br>驱动器可用空间总量: " + drive.TotalFreeSpace.ToString());
Response.Write("<br>驱动器总空间: " + drive.TotalSize.ToString());
Response.Write("<br>卷标: " + drive.VolumeLabel);
```

(4) 按【Ctrl+F5】组合键运行网页，结果如图 8.1 所示。

使用 DriveInfo 的 GetDrives() 静态方法，还可以列举本地文件系统上的所有驱动器。

【例 8-2】 使用 DriveInfo 列举本地文件系统的所有驱动器信息。

设计步骤如下：

(1) 在网站 “Chapter8” 中添加一个网页 “example8-2.aspx”，切换到设计视图，拖放一个 TreeView 控件到页面上。

(2) 打开 example8-2.aspx.cs 代码页，添加命名空间：

```
using System.IO;
```

(3) 在 Page\_Load 方法体内添加如下代码：

```
if (!Page.IsPostBack)
{
    foreach (DriveInfo drive in DriveInfo.GetDrives())
    {
        TreeNode node = new TreeNode();
        node.Value = drive.Name;
        if (drive.IsReady) //在访问驱动器之前要先检查 IsReady 属性
```

```
node.Text = drive.Name + " : (剩余空间: " + drive.AvailableFreeSpace + "字节)";
else
    node.Text = drive.Name + " : (未就绪)";
TreeView1.Nodes.Add(node);
}
}
```

(4) 按【Ctrl+F5】组合键运行网页，结果如图 8.2 所示。

上面的代码使用了 DriveInfo 的 GetDrives()静态方法,列举了本地文件系统上的所有驱动器,并把每个驱动器作为一个根节点添加到 TreeView 控件中。

在执行代码时,有些驱动器如移动式驱动器、网络驱动器等并不总是可以访问的,驱动器的 IsReady 属性是一个只读属性,用于测试该驱动器是否可以访问,在访问驱动器之前应该首先检查它。

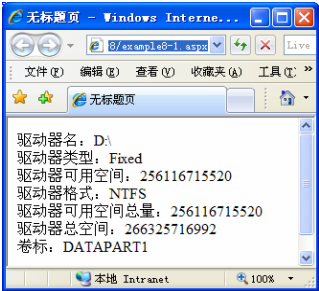


图 8.1 example8-1 网页的运行结果

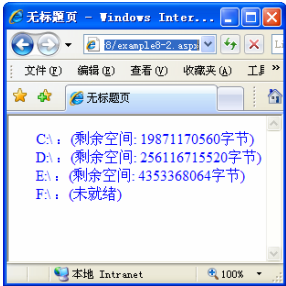


图 8.2 example8-2 网页的运行结果

8.1.2 文件夹操作

对文件夹的操作可以使用 Directory 或 DirectoryInfo 类。Directory 类中包含许多静态方法,可以直接使用该类的静态方法创建、移动和删除文件夹。表 8.3 分别列出了 Directory 类的主要成员。

表 8.3 Directory 类的主要成员

成 员 名	说 明
CreateDirectory()	创建指定路径中的所有文件夹
Delete()	删除指定的文件夹
Move()	将文件或文件夹及其内容移到新位置
Exists()	确定给定路径是否引用磁盘上的现有文件夹
GetCreationTime()	获取文件夹的创建日期和时间
GetCurrentDirectory()	获取应用程序的当前工作文件夹
SetCurrentDirectory()	将应用程序的当前工作文件夹设置为指定的文件夹
GetParent()	检索指定路径的父文件夹,包括绝对路径和相对路径
GetDirectories()	获取指定文件夹中子文件夹的名称
GetFiles()	返回指定文件夹中的文件的名称

**【例 8-3】** 使用 `Directory` 类的静态方法创建文件夹和删除文件夹。

设计步骤如下：

(1) 在网站“Chapter8”中添加一个网页“example8-3.aspx”，切换到设计视图，拖放 1 个 `DropDownList` 控件、1 个 `TextBox` 控件、2 个 `Button` 控件和 1 个 `RequiredFieldValidator` 验证控件到页面上。

(2) `RequiredFieldValidator` 控件的 `ErrorMessage` 属性设置为“请输入文件夹名”，`ControlToValidate` 的属性选择 `TextBox1`。`Button1` 和 `Button2` 的 `Text` 属性分别设置为“创建文件夹”和“删除文件夹”。

(3) 打开 `example8-3.aspx.cs` 代码页，添加命名空间：

```
using System.IO;
```

(4) 在 `Page_Load` 方法体内添加如下代码：

```
if (!Page.IsPostBack)                                //如果第一次加载页面则执行下面代码
{
    foreach (DriveInfo drive in DriveInfo.GetDrives())
    {
        if (drive.IsReady)
        {
            DropDownList1.Items.Add(drive.Name); //将盘符名加载到DropDownList1 中
        }
    }
}
```

(5) 切换到设计视图，双击 `Button1`，给 `Button1` 添加事件代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
    //创建文件夹的路径
    string pathDirectory = DropDownList1.SelectedValue + TextBox1.Text;
    Directory.CreateDirectory(pathDirectory);    //在指定路径下创建文件夹

    if (Directory.Exists(pathDirectory))          //如果创建成功则输出创建信息
    {
        Response.Write(TextBox1.Text + "文件夹创建成功，创建时间为：" +
            Directory.GetCreationTime(pathDirectory).ToString());
    }
}
```

(6) 切换到设计视图，双击 `Button2`，给 `Button2` 添加事件代码：

```
protected void Button2_Click(object sender, EventArgs e)
{
    //删除文件夹的路径
    string pathDirectory = DropDownList1.SelectedValue + TextBox1.Text;
    if (Directory.Exists(pathDirectory))          //如果存在此文件夹则执行下面代码
    {
        Directory.Delete(pathDirectory);    //删除文件夹
        Response.Write(TextBox1.Text + "文件夹删除成功");
    }
}
```

```
    }  
}
```

(7) 按【Ctrl+F5】组合键运行网页，选择盘符，输入文件夹名，单击【创建文件夹】按钮，结果如图 8.3 所示。选择盘符，输入文件夹名，单击【删除文件夹】按钮，结果如图 8.4 所示。

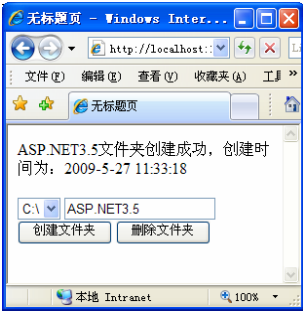


图 8.3 创建文件夹



图 8.4 删除文件夹

DirectoryInfo 类对象可以表示一个特定的文件夹，读者可以在该对象上执行与 Directory 类相同的操作，还可以使用它列举子文件夹和文件。DirectoryInfo 类的主要成员见表 8.4。

表 8.4 DirectoryInfo 类的主要成员

成 员 名	说 明
Create()	创建文件夹
CreateSubdirectory()	在指定路径中创建一个或多个子文件夹
Delete()	从路径中删除 DirectoryInfo 及其内容
MoveTo	将 DirectoryInfo 实例及其内容移动到新路径
GetDirectories()	返回当前文件夹的子文件夹
GetFiles()	返回当前文件夹的文件列表

**【例 8-4】** 列举系统中盘符下的文件夹。

设计步骤如下：

(1) 在网站“Chapter8”中添加一个网页“example8-4.aspx”，切换到设计视图，拖放 1 个 TreeView 控件到页面上。

(2) 打开 example8-4.aspx.cs 代码页，添加命名空间：

```
using System.IO;
```

(3)添加一个方法 LoadDirectories。在指定路径“path”下查找出所有的文件夹作为“parent”子节点。

```
private void LoadDirectories(TreeNode parent, string path)  
{  
    //根据楼价实例化一个对象 directory  
    DirectoryInfo directory = new DirectoryInfo(path);  
    try  
    {
```

```

        foreach (DirectoryInfo d in directory.GetDirectories())//遍历 directory 子文件夹
        {
            TreeNode node = new TreeNode(d.Name, d.FullName);
            parent.ChildNodes.Add(node);
        }
    }
    catch (System.UnauthorizedAccessException e)           //异常处理
    { parent.Text += " (拒绝访问," + e.Message + ")"; }
    catch (System.IO.IOException e)
    { parent.Text += " (未知错误:" + e.Message + ")"; }
}

```

(4) 在 Page\_Load 方法体内添加如下代码:

```

if (!Page.IsPostBack)
{
    foreach (DriveInfo drive in DriveInfo.GetDrives())      //遍历系统下的盘符
    {
        TreeNode node = new TreeNode();
        node.Value = drive.Name;
        if (drive.IsReady)
        {
            node.Text = drive.Name + " - (可用空间有:" + drive.AvailableFreeSpace + "字节)";
            LoadDirectories(node, drive.Name); //调用 LoadDirectories 方法
        }
        else
        {
            node.Text = drive.Name + " - (未就绪)";
            TreeView1.Nodes.Add(node);
        }
    }
    TreeView1.CollapseAll(); //关闭树中的每个节点
}

```

(5) 按【Ctrl+F5】组合键运行网页，结果如图 8.5 所示。

上面的代码不仅列举了本地文件系统中的驱动器，而且还添加了浏览系统的文件夹结构的功能。使用 DirectoryInfo 类的 GetDirectories()方法，访问系统驱动器下的文件夹，并将它们逐一添加到 TreeView 控件中，如图 8.5 所示。

### 8.1.3 文件操作

使用 Directory 类和 DirectoryInfo 类可以很方便地管理文件夹，而 File 类和 FileInfo 类则可以进一步对文件夹内的文件进行操作。与 Directory 类相似，File 类提供了用于创建、复制、移动、删除和打开文件的静态方法，并协助创建 FileStream 对象。表 8.5 列出了 File 类的主要方法。

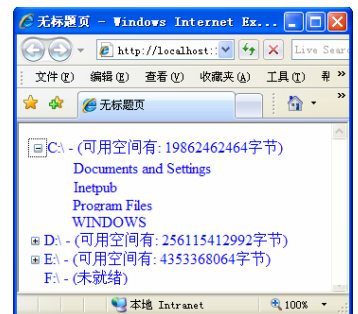


图 8.5 example8-4 网页的运行结果

表 8.5 File 类的主要方法

方 法 名	说 明
Create()	已重载，在指定路径中创建文件
Copy()	将现有文件复制到新文件
Move()	将指定文件移到新位置，并提供指定新文件名的选项
Delete()	删除指定的文件，如果指定的文件不存在，并不引发异常
Exists()	确定指定的文件是否存在
GetLastWriteTime()	返回上次写入指定文件或文件夹的日期和时间
OpenWrite()	打开现有文件以进行写入
ReadAllText()	打开文本文件，将文件的所有行读入一个字符串，然后关闭该文件
AppendAllText()	打开文件，向其中追加指定的字符串，然后关闭该文件。如果文件不存在则创建一个文件

**【例 8-5】** 利用 File 类在网站中创建文本文件并且在所创建的文件中添加内容。  
设计步骤如下：

(1) 在网站“Chapter8”中添加一个网页“example8-5.aspx”，切换到设计视图，添加 1 个 4 行 2 列的表，在表中放入 3 个 TextBox 控件、2 个 Button 控件和 1 个 RequiredFieldValidator 验证控件，添加一些提示，如图 8.6 所示。

(2) TextBox2 和 TextBox3 的 TextMode 属性设置为“MultiLine”，RequiredFieldValidator1 的 ErrorMessage 设置为“不能为空值”，ControlToValidate 设置为“TextBox1”，Button1 和 Button2 的 Text 分别设置为“添加”和“删除”。

(3) 打开 example8-5.aspx.cs 代码页，添加命名空间：

```
using System.IO;
```

(4) 切换到设计视图，双击 Button1，添加 Button1 的 Click 事件代码，如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    string b = Request.PhysicalApplicationPath;           //获取网站根目录的物理路径
    File.AppendAllText(b + TextBox1.Text.Trim(), TextBox2.Text);
    string a= File.ReadAllText(b+TextBox1.Text.Trim());
    TextBox3.Text = a;
}
```

(5) 切换到设计视图，双击 Button2，添加 Button2 的 Click 事件代码，如下所示：

```
protected void Button2_Click(object sender, EventArgs e)
{
    string b = Request.PhysicalApplicationPath;
    File.Delete(b+TextBox1.Text.Trim());
    Response.Write("<script>alert('文件删除成功！')</script>");
}
```

(6) 按【Ctrl+F5】组合键运行网页，输入文件名，输入要添加的内容，结果如图 8.7 所示。



图 8.6 example8-5 网页设计



图 8.7 example8-5 网页的运行结果

与 DirectoryInfo 类相类似，FileInfo 类提供与 File 类相似的功能，但是必须使用该类的对象来进行特定文件的相关操作。FileInfo 对象中记录了文件的文件名、大小、创建时间等属性，因此显示文件夹内所有文件信息的最简单的方法是将一个 FileInfo 对象数组绑定到 GridView 上，可以使用 DirectoryInfo 类的 GetFiles()方法来获得 FileInfo 对象数组。如果只需要获取文件夹中的文件名列表，则可以使用 Directory 类的 GetFiles()方法获得文件夹中文件名的字符串数组。表 8.6 列出了 FileInfo 类的主要方法。

表 8.6 FileInfo 类的主要方法

方 法 名	说 明
Create()	在指定路径中创建文件，返回 FileStream 对象用于写入
CreateText()	在指定路径中创建写入新文本文件的 StreamWriter
Delete()	永久删除文件
MoveTo()	将指定文件移到新位置，并提供指定新文件名的选项
CopyTo()	将现有文件复制到新文件
Open()	用各种读/写访问权限和共享特权打开文件
OpenRead()	创建只读 FileStream
OpenText()	创建只读的 StreamReader，用于文本文件的读取
OpenWrite()	创建只写的 FileStream

【例 8-6】 利用 FileInfo 显示站点中的所有文件。

设计步骤如下：

- (1) 在网站“Chapter8”中添加一个网页“example8-6.aspx”，切换到设计视图，添加 1 个 Button 控件。
- (2) Button1 的 Text 设置为“显示网站下的所有文件”。
- (3) 打开 example8-6.aspx.cs 代码页，添加命名空间：

```
using System.IO;
```

- (4) 切换到设计视图，双击 Button1，添加 Button1 的 Click 事件代码，如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    string b = Request.PhysicalApplicationPath;
```



```
string s = "此站点下包含的文件有: <br>";
DirectoryInfo directory = new DirectoryInfo(b);
foreach (FileInfo f in directory.GetFiles())
{
    s += f.Name + "<br>";
}
Response.Write(s);
}
```

(5) 按【Ctrl+F5】组合键运行网页，单击按钮，结果如图 8.8 所示。

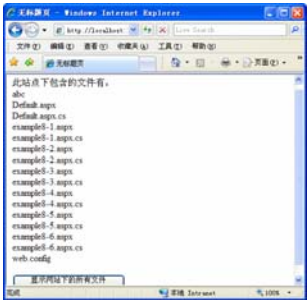


图 8.8 example8-6 网页的运行结果

8.1.4 使用路径

在操作文件夹和文件的时候，经常需要指明该文件夹或文件所在的路径，有时还需对路径进行处理。由于路径信息是以普通的字符串形式存储的，所以将组成路径的各个子串连接在一起判断文件的扩展名，从路径名中提取出文件名等都是开发人员经常需要完成的工作。.NET Framework 提供了专门的 Path 类来处理与路径相关的任务。在 System.IO.Path 类中有许多静态方法，可以很好地处理路径。表 8.7 中列出了 Path 类的主要方法。

表 8.7 Path 类的主要方法

方 法 名	说 明
ChangeExtension( )	更改路径字符串的扩展名
GetDirectoryName( )	返回指定路径字符串的文件夹信息
GetExtension( )	返回指定路径字符串的扩展名
GetFileName( )	返回指定路径字符串的文件名和扩展名
GetFileNameWithoutExtension( )	返回不具有扩展名的指定路径字符串的文件名
GetFullPath( )	返回指定路径字符串的绝对路径
GetPathRoot( )	获取指定路径的根文件夹信息
HasExtension( )	确定路径是否包括文件扩展名

【例 8-7】 利用 Path 类输出运行网页的路径信息到页面上。  
设计步骤如下：

- (1) 在网站“Chapter8”中添加一个网页“example8-7.aspx”。
- (2) 打开 example8-7.aspx.cs 代码页，添加命名空间：

```
using System.IO;
```

(3) 在 Page\_Load 方法体内添加代码，如下所示：

```
string filename = Request.PhysicalPath;           //获取 example8-7.aspx 的物理路径
if(Path.HasExtension(filename))                   //确定路径是否包括文件扩展名
{
    Response.Write("根路径: " + Path.GetPathRoot(filename));
    Response.Write("<br>绝对路径: " + Path.GetFullPath(filename));
    Response.Write("<br>文件夹名称: " + Path.GetDirectoryName(filename));
    Response.Write("<br>文件名: " + Path.GetFileName(filename));
    Response.Write("<br>文件名(不包含扩展名):"+ Path.GetFileNameWithoutExtension
(filename));
    Response.Write("<br>文件扩展名: " + Path.GetExtension(filename));
}
```

(4) 按【Ctrl+F5】组合键运行网页，结果如图 8.9 所示。

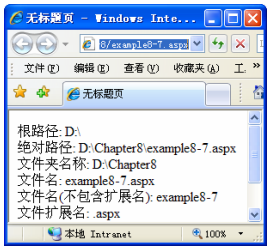


图 8.9 example8-7 网页的运行结果

## 8.2 文件读写操作

在前面一节中，介绍了如何管理服务器的本地文件系统，本节将介绍如何使用 .NET Framework 读写文件。Windows 文件系统和 UNIX 文件系统都是流文件系统，简单的说就是将文件处理为字符流或二进制流，所以对文件的读写就是读取字符流或二进制流。在 .NET Framework 中，对文件的读写操作非常简单，因为它使用读写 I/O 数据的通用模型，无论数据源是什么，都可以使用相同的代码。该模型的核心是 Stream 类和 Reader/Writer 类。图 8.10 显示了 .Net Framework 中基本的 I/O 流模型。

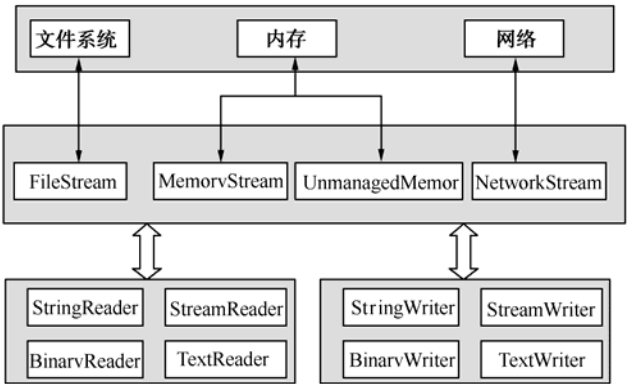


图 8.10 .Net Framework 中基本的 I/O 流模型

Stream 类提供了读写 I/O 数据的基本功能，因为它是一个抽象类，所以在使用时应使用其派生类。表 8.8 列举了常用的 Stream 类的派生类。

表 8.8 Stream 类的派生类

类 名	说 明
FileStream	对文件系统上的文件进行读取、写入、打开和关闭操作，对其他与文件相关的操作系统句柄进行操作，如管道、标准输入和标准输出。读写操作可以指定为同步或异步操作
MemoryStream	创建以内存而不是磁盘或网络连接作为支持存储区的流
BufferedStream	可以向另一个 Stream(如 NetworkStream)添加缓冲的 Stream (FileStream 内部已具有缓冲, MemoryStream 无须缓冲) 流，在缓冲区内执行操作，可以提高效率
NetWorkStream	提供用于网络访问的基础数据流
GZipStream	提供用于压缩和解压缩流的方法和属性
DeflateStream	提供用于使用 Deflate 算法压缩和解压缩流的方法和属性

8.2.1 使用FileStream类读写文件

【例 8-8】 使用 FileStream 类读写文本文件。

设计步骤如下：

- (1) 在网站“Chapter8”中添加 1 个网页“example8-8.aspx”、2 个文本文件，2 个文本文件分别命名为“FileIn.txt”和“FileOut.txt”。
- (2) 在 FileIn.txt 文件中添加文本“功能强大”，在 FileOut.txt 文件中添加文件“c#”。
- (3) 打开 example8-8.aspx.cs 代码页，添加命名空间：

```
using System.IO;
```

(4) 在 Page\_Load 方法体内添加代码，如下所示：

```
//新建 FileStream 对象，以 open 方式打开
FileStream fs = new FileStream(Server.MapPath("FileIn.txt"), FileMode.Open);
byte[] data = new byte[fs.Length];
fs.Read(data, 0,(int)fs.Length);
fs.Close(); //关闭 fs 并释放与之关联的所有资源
//新建 FileStream 对象，以 Append 方式打开进行写入
fs = new FileStream(Server.MapPath("FileOut.Txt"),FileMode.Append, FileAccess.Write);
fs.Write(data, 0,(int)data.Length);
fs.Flush(); //清除 fs 的缓冲区
fs.Close();
```

(5) 按【Ctrl+F5】组合键运行网页，打开 FileOut.txt 文件，结果如图 8.11 所示。

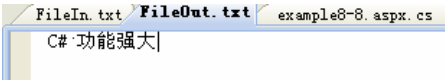


图 8.11 打开 FileOut.txt 文件后的部分截图

程序分析：

(1) 首先创建 FileStream 对象。FileStream 构造函数的第一个参数是要访问的文件的路径，因此，在后面的代码中可以使用该对象对 FileIn.txt 文件进行操作。第二个参数是文件打开时要使用的访问类型，本例中在创建 FileStream 对象时打开了 FileIn.txt 文件。

(2) 接下来创建一个字节数组 data。其长度由 FileStream 对象的长度决定，使用 FileStream 对象的 Read 方法将文件流数据读取到 data 数组中。

(3) 必须显式地关闭 FileStream，以释放它们占用的资源。如果没有显式关闭或者关闭失败，则有可能导致内存泄漏，或者导致其他用户和应用程序无法访问该资源。

在打开文件的时候除了要指定对文件的访问方式外，还需要考虑文件的访问控制权限、文件共享及其他一些相关问题。FileStream 构造函数的一系列重载函数可以显式地指定如何使用文件。IO 命名空间包含了 4 个枚举成员来帮助控制 FileStream 对文件的访问。

- FileMode: 用于控制是否对文件执行覆盖、创建、打开等操作，或执行这些操作的组合。
- FileAccess: 用于控制对文件进行读访问、写访问或读/写访问。
- FileOptions: 文件访问的其他控制选项，如随机/顺序访问、文件加密、异步文件写入等。
- FileShare: 用于控制正在使用的文件是否可以被其他用户或程序共享。

在【例 8-8】中第二次创建的 FileStream 对象以追加写入的方式打开了 FileOut.txt 文件。然后使用 Write 方法，将刚刚读取出来的字节数组写入 FileStream 缓冲区，再使用 Flush 方法通知 FileStream 将缓冲区的内容写入 FileOut.txt 中。最后关闭 FileStream，释放它所占用的资源。

需要注意的是，在本例中 Flush 方法是可选的，因为 Close 方法会在内部调用 Flush，将缓冲区的内容输出到文件。但由于 Flush 方法不会像 Close 方法一样释放 FileStream 资源，因此使用它可以执行多个写入操作。

可以使用相似的技术读写任意的 Stream 派生类，如将数据写入 MemoryStream、NetworkStream 等，读者可以自行试一试。

8.2.2 使用Reader/Writer类读写文件

虽然使用 FileStream 类读写文件非常简单，但它把所有数据都作为字节流看待。在程序开发过程中，开发人员更希望能够直接处理各种类型的数据。.NET Framework 提供了许多 Reader 类和 Writer 类，它们都根据特定的规则进行设计，把对流的读写操作封装起来，这样开发人员就可以集中精力处理数据。表 8.9 列出了常用的 Reader 类和 Writer 类。

表 8.9 常用的 Reader 类和 Writer 类

类 名	说 明
TextReader	StreamReader 和 StringReader 的抽象基类。用于读取 Unicode 字符
TextWriter	StreamWriter 和 StringWriter 的抽象基类。用于输出 Unicode 字符
StreamReader	TextReader 类的派生类，用于从字节流中读取字符
StreamWriter	TextWriter 类的派生类，用于把字符写入字节流中
StringReader	TextReader 类的派生类，用于从 Strings 中读取字符
StringWriter	TextWriter 类的派生类，用于向 String 中写入字符
BinaryReader	从流中把基本数据类型读取为二进制值

### 【例 8-9】 使用 StreamReader 类读取文本文件。

设计步骤如下：

- (1) 在网站 Chapter8 中添加一个网页 “example8-9.aspx”，切换到设计视图，添加 1 个 TextBox 控件和 1 个 Button 控件。
- (2) TextBox1 的 TextMode 属性设置为 “MultiLine”，Button1 的 Text 设置为 “输出到页面”。
- (3) 打开 example8-9.aspx.cs 代码页，添加命名空间：

```
using System.IO;
```

- (4) 切换到设计视图，双击 Button1，添加 Button1 的 Click 事件代码，如下所示：

```
string text = TextBox1.Text;
//新建 StreamWriter 对象，打开 MyText.txt 文件，如果没有 MyText.txt 文件则创建
StreamWriterstrwriter=newStreamWriter(File.Open(Server.MapPath("MyText.txt"),
FileMode.Append));
strwriter.Write(text);
strwriter.Close();
//新建 StreamReader 对象，打开 MyText.txt 文件，将内容读入到 tmp 字符串
StreamReaderstrreader=newStreamReader(File.Open(Server.MapPath("MyText.txt"),
FileMode.Open));
string tmp;
while ((tmp = strreader.ReadLine()) != null)
    Response.Write(tmp + "<br>");
strreader.Close();
```

- (5) 按【Ctrl+F5】组合键运行网页，在文本框中输入“使用 StreamReader 读取文本文件”，单击按钮，结果如图 8.12 所示。

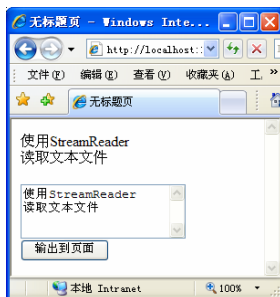


图 8.12 example8-9 网页的运行结果

在创建 StreamReader 时，必须把一个已有的流实例作为构造函数的参数。StreamReader 就把这个流作为自己的数据源。同样，StreamWriter 的创建也需要一个已有流作为写入的目的流。另外，StreamReader 无须处理字节数组，它提供了多种方法读取数据，例如，使用 ReadToEnd()方法可以读取整个文件，使用 ReadLine()方法可以读取一行，使用 Read()方法可以读取一个字符。

8.2.3 文件压缩

.NET 2.0 Framework 以上版本引入了名称空间 System.IO.Compression。该名称空间包含的 GZipStream 类和 DeflateStream 类可以用于压缩和解压缩数据。这两个类都派生于 Stream 类，因此其用法和本章前面介绍的其他 Stream 类的操作相同。

1. 文件的压缩


【例 8-10】 使用 GZip 压缩【例 8-9】所创建的 MyText.txt 文件。  
设计步骤如下：

- (1) 在网站“Chapter8”中添加一个网页“example8-10.aspx”。
- (2) 打开 example8-10.aspx.cs 代码页，添加命名空间：

```
using System.IO;
using System.IO.Compression;
```

- (3) 在 Page\_Load 方法体内添加代码，如下所示：

```
// 读取需要压缩的文件
string fileName = Server.MapPath("MyText.txt");
FileStream inStream = File.OpenRead(fileName);
byte[] buffer = new byte[inStream.Length];
inStream.Read(buffer, 0, buffer.Length);
inStream.Close();
// 创建输出文件
FileStream outputStream = File.Create(Path.ChangeExtension(fileName, "zip"));
// 压缩输入流，并把它写入到输出流中
GZipStream gzip = new GZipStream(outputStream, CompressionMode.Compress);
gzip.Write(buffer, 0, buffer.Length);
gzip.Close();
outputStream.Close();
```

- (4) 按【Ctrl+F5】组合键运行网页→打开“解决方案资源管理器”→单击“刷新”按钮，结果如图 8.13 所示。

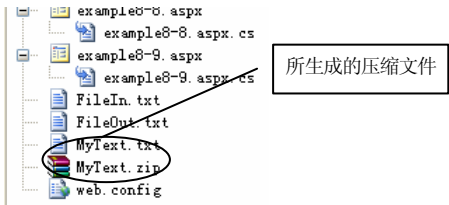


图 8.13 （例 8-10）解决方案资源管理器部分截图

程序分析：

- (1) 程序中创建了两个 FileStream 对象和一个 GzipStream 对象。需要压缩的真实的流被读取到 inStream 中，由 GzipStream 对象 gzip 对 inStream 中的数据进行压缩后，将压缩后

的数据传送到 outStream 中。

(2) 在构造 GZipStream 流对象时需要两个参数，一个参数用于指定准备写入压缩数据的流，另一个参数用于指明是要进行压缩还是解压缩。

(3) 在写压缩数据时，应使用压缩流的 Write()方法，而不是 FileStream 的 Write()方法。

## 2. 文件的解压缩

解压缩的操作与压缩的操作基本相同，区别在于把“CompressionMode.Compress”改为“CompressionMode.Decompress”。

**【例 8-11】** 使用 GZip 解压缩【例 8-10】所生成的 MyText.Zip 的压缩文件。  
设计步骤如下：


(1) 在网站“Chapter8”中添加一个网页“example8-11.aspx”。

(2) 打开 example8-11.aspx.cs 代码页，添加命名空间：

```
using System.IO;
using System.IO.Compression;
```

(3) 在 Page\_Load 方法体内添加代码，如下所示：

```
//读取压缩文件
string fileName = Server.MapPath("MyText.zip");
FileStream inStream = File.OpenRead(fileName);
// 创建 GZipStream 对象
GZipStream gzip = new GZipStream(inStream, CompressionMode.Decompress);
byte[] buffer = new byte[inStream.Length];
gzip.Read(buffer, 0, buffer.Length);
inStream.Close();
string s1 = Path.GetDirectoryName(fileName); //获取指定路径字符串的文件夹信息
//获取不具有扩展名的指定路径字符串的文件名
string s2 = Path.GetFileNameWithoutExtension(fileName).ToString();
string s3=s1+"@"+"s2+"_解压.txt"; //合并成输出文件路径和文件名
FileStream outStream = File.Create(s3); //创建输出文件流
outStream.Write(buffer, 0, buffer.Length); //将解压缩后的数据写入文件流
gzip.Close(); //关闭当前流并释放资源
outStream.Close(); //关闭当前流并释放资源
```

(4) 按【Ctrl+F5】组合键运行网页→打开“解决方案资源管理器”→单击“刷新”按钮，结果如图 8.14 所示。

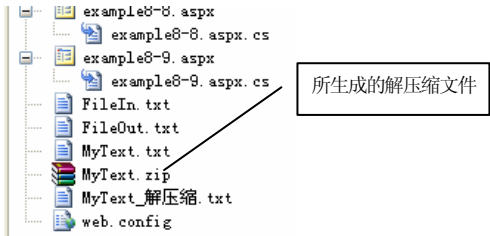


图 8.14 （例 8-11）解决方案资源管理器部分截图

## 8.3 文件上传

网络应用程序中常常需要交换各种信息，而文件上传是重要的信息交换方式之一。ASP.NET 2.0 以上版本提供了一个 **FileUpload** 控件用于将文件上传到 Web 服务器。Web 服务器接收到上传文件后，可以通过程序对它进行处理，或者忽略它，或者保存到后端数据库或服务器文件夹中。

**FileUpload** 控件使用户能够上传图片、文本文件或其他文件。它包含一个文本框控件和一个按钮控件，用户可以在文本框中输入希望上传到服务器的文件的完整路径，也可以通过【浏览】按钮浏览并选择需要上传的文件。出于安全方面的考虑，不能将文件名预先加载到 **FileUpload** 控件中。

用户选择好要上传的文件后，**FileUpload** 控件并不会自动上传文件，而必须提交页面并进行相应的处理。当用户已选定要上传的文件并提交页面时，该文件将作为请求的一部分上传。上传后的文件将被完整地缓存在服务器内存中。在访问该文件前首先需要测试 **FileUpload** 控件的 **HasFile** 属性，检查该控件是否有上传的文件。如果 **HasFile** 返回 **True**，就可以调用 **HttpPostedFile** 对象的 **SaveAs** 方法将上传的文件保存到服务器指定的位置。另外，还可以使用 **HttpPostedFile** 对象的 **InputStream** 属性，以字节数组或字节流的形式管理已上传的文件。

**FileUpload** 控件每次只能够上传一个文件，这些文件可以是普通的图片、文本，也可以是脚本、可执行文件等。为了防止有害的文件被上传，在上传文件之前，最好能够设置一定的规则对文件予以筛选，如可以检测文件的扩展名，也可以使用验证控件对要上传的文件名进行正则表达式检查。

**【例 8-12】** 使用 **FileUpload** 控件上传图片并显示在页面上。

设计步骤如下：

(1) 在网站“Chapter8”中添加一个网页“example8-12.aspx”。切换到设计视图，从工具箱中拖放 1 个 **FileUpload** 控件、1 个 **Button** 控件、1 个 **Lable** 控件和 1 个 **Image** 控件到页面上。

(2) 打开“解决方案资源管理器”，添加 1 个命名为“**Images**”的文件夹，打开 example8-12.aspx.cs 代码页，添加命名空间：

```
using System.IO;
```

(3) 切换到设计视图，双击 **Button1**，添加 **Click** 事件代码，如下所示：

```
Boolean fileOK = false;
String path = Server.MapPath("~/Images/");           //Images 文件夹的物理路径
if (FileUpload1.HasFile)                             //判断是否有文件被上传
{
    //检查上传文件的扩展名是否在许可范围内
    String fileExtension =
        Path.GetExtension(FileUpload1.FileName).ToLower(); //获得文件扩展名并转换
                                                            为小写
    String[] allowedExtensions = { ".bmp", ".jpeg", ".jpg", ".gif" }; //指定文件后缀名
    for (int i = 0; i < allowedExtensions.Length; i++)
    {
        if (fileExtension == allowedExtensions[i])
        {
            fileOK = true;
        }
    }
}
```



```

    }
}
if (fileOK)

//如果上传的文件后缀
//名在指定的类型中，则执
//行下面代码

{
    try
    {
        //保存上传后的文件到指定的文件夹中
        FileUpload1.PostedFile.SaveAs(path + FileUpload1.FileName);
        Image1.ImageUrl = @"~/Images/" + FileUpload1.FileName;
        Label1.Text = "文件" + FileUpload1.FileName + "被成功上传";
    }
    catch
    { Label1.Text = "文件上传失败! "; }
}
else
{ Label1.Text = "错误的文件类型"; }

```

(4) 按【Ctrl+F5】组合键运行网页→单击【浏览】按钮→选择图片→单击【上传】按钮，结果如图 8.15 所示。

可以通过以下三种方式获取上传文件的详细内容和属性。

- 通过 FileUpload 控件的 FileBytes 属性，获取上传文件的字节数组。

- 通过 FileContent 属性，获取指向上传文件的 Stream 对象。使用该属性可以以字节方式访问文件内容。

- 通过 PostedFile 属性，获取一个与上传文件相关的 HttpPostedFile 对象。可以使用该属性访问文件的其他属性，ContentLength 属性获取文件的长度，ContentType 属性获取文件的 MIME 内容属性。

注意：

(1) 可上传的最大文件的大小取决于 MaxRequestLength 配置设置的值，默认大小为 4096KB。如果用户试图上传超过最大文件大小的文件，上传就会失败，这样可以防止通过发送大量的请求攻击服务器的行为。

(2) 上传的文件将被保存到服务器的某个文件夹中，该文件夹是否允许用户访问和写入是影响上传是否成功的决定性因素，因此必须为存储上传文件的文件夹授予写入权限。

## 习 题

1. Directory 和 DirectoryInfo 类有什么区别和联系？File 和 FileInfo 类有什么区别和联系？
2. 简述.NET 框架的基本 I/O 流模型及其工作原理。
3. 简述对文件进行压缩和解压缩的过程。



图 8.15 (例 8-12) 上传文件后的页面

## 第 9 章 ASP.NET 高级技术

在开发较大型的 Web 应用程序时，往往需要实现一些特殊的功能（如安全等）、灵活的配置及性能的优化，以应对大量用户频繁的访问。本章将介绍 ASP.NET 的几种高级技术，利用这些技术可以方便、高效地开发高级的 ASP.NET 程序，灵活地配置 Web 应用程序，有效地提高网站的可伸缩性。

### 9.1 ASP.NET 配置

ASP.NET 使用一个基于 XML 文件的配置系统，它更灵活、访问性更高、更易使用。

#### 9.1.1 ASP.NET 配置概述

每当我们建立一个应用程序时，通常都需要利用一些描述应用程序行为和设置的信息，也就是通常所说的配置信息，如数据库连接字符串、超时值、如何记录错误、如何维持状态等。ASP.NET 采用了一个易操作且功能强大的基于 XML 的配置系统，不仅可以在设计阶段定义配置设置，也可以在运行期间定义配置设置，可以随时添加或改变配置信息，可以直接激活定义的新配置设置，而不会使服务器的效率有任何损失。

ASP.NET 采用基于 XML 的配置文件，易于定制。这些文件可以通过在文本编辑器中编辑来配置 ASP.NET 的任何组件。

ASP.NET 配置系统支持如下 2 类配置文件。

##### （1）服务器配置——machine.config 文件

服务器配置信息存储在一个名为 machine.config 的文件中。这个文件描述了所有 ASP.NET Web 应用程序所用的默认设置。该文件被安装在服务器的如下目录中：

```
[WinNT]\Microsoft.NET\Framework\[version]\Config // version 指的是 .NET 框架版本
```

##### （2）应用程序配置——web.config 文件

应用程序配置信息存储在一个名为 web.config 的文件中。该文件描述了一个单独的 ASP.NET 应用程序的设置信息。一个服务器可以有多个 web.config 文件，每个文件都放在应用程序的根目录下或者在应用程序内部的子目录中。每个子文件夹中的 web.config 文件会继承或重写父文件夹的设置。

#### 9.1.2 配置文件的结构

##### 1. 配置文件的基本规则

ASP.NET 的配置文件是基于 XML 的文件，它必须符合基本的 XML 规则。

（1）这些文件必须有一个唯一的根元素，machine.config 和 web.config 的根元素都是

<configuration>。

(2) 定义的任何元素必须包括在一对起始和结束标识符内，如<start>和</start>，这些标识符是区分大小写的。

(3) 任何属性、关键字或值都必须包括在双引号中。例如：

```
<add key="data"/>
```

(4) 元素必须嵌套，不能重叠。

## 2. 配置文件的格式

在配置文件的根元素<configuration>之下有 2 个主要部分。

### (1) 声明部分

声明部分用<configSections>标识符来界定，在其中用<section name="data">元素来定义一个类，我们称其为配置项处理程序，用来解释配置数据的含义。如果配置项处理程序在 machine.config 中声明过，则对于所有应用程序，它只需声明一次（因为应用程序可以继承 machine.config 中的设置）。

### (2) 设置部分

设置部分用<[sectionSettings]>标识符来界定，该标识符是在声明部分定义的，我们称它为配置项设置，它定义了一个特定选项的实际设置。配置项设置可以覆盖从 machine.config 继承的设置。

配置文件的声明部分和设置部分都可以包含在一个<sectionGroup>中。在配置文件中<sectionGroup>起着组织的作用，它可以将配置组织成组，例如，<system.web>项组用来标识配置文件中的 ASP.NET 的专门区域。

## 9.1.3 常用配置

### 1. 一般配置

在配置文件中，一般配置部分包含一般的应用配置设置，放在<httpRuntime>标识符中，通常它们被包含在<system.web>项组中。下面是一个例子：

```
<configuration>
  <system.web>
    <httpRuntime executionTimeout="90" maxRequestLength="4096"/>
  </system.web>
</configuration>
```

上述配置中的参数说明如下。

(1) executionTimeout：用来定义请求的超时时间，单位为秒。

(2) maxRequestLength：指定请求的最大长度（KB），默认为 4MB，该值的设置可以限制用户请求的内容大小，例如，用户要上传文件，通过该设置可以限制上传文件的大小。

## 2. 连接字符串配置

从 ASP.NET 2.0 以后, 数据库的连接字符串存储在<connectionStrings>段中。下面的例子存储了 XSCJ 数据库的连接字符串:

```
<configuration>
  <connectionStrings>
    <add name="XSCJConnectionString" connectionString="DataSource=localhost;Initial Catalog=XSCJ;Integrated Security=True" providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

使用上面的配置后, 如果数据库位置改变或更换了数据库, 则只需修改配置文件的连接字符串设置, 无须修改程序。

## 3. 应用程序配置

在配置文件中, 应用程序配置部分包含应用配置信息, 放在<appSettings>标识符中。通过应用程序配置部分, 我们可以将应用程序配置细节存储到配置文件中, 而不需要编写自己的配置项处理程序。这里声明的键/值对, 用于填充一个可以在应用程序中访问的表, 这是一个优点, 因为配置文件不能通过 HTTP 访问, 所以它可以禁止非法访问关键的应用程序参数。下面的例子存储了应用程序的一个参数 AppKey:

```
<configuration>
  <appSettings>
    <add key="AppKey" value="importantkey" />
  </appSettings>
</configuration>
```

关键字“AppKey”添加到表中, value 的值也添加进去, 我们可以在应用程序中访问这些信息, 语句如下所示:

```
ConfigurationSettings.AppSettings["AppKey"]
```

## 4. 定制错误

尽管应用程序在发布前都进行了全面的测试, 但错误仍会发生。当应用程序中出现一个运行时或设计时的错误时, ASP.NET 就会显示一个非常有用的错误页面, 这样便于开发人员调试代码, 但显然我们并不希望将这类错误的详细信息都显示给终端用户。最好能修改应用程序处理错误的方式, 当错误产生时把用户重定向到其他页面。

可以使用 web.config 配置文件中的<customErrors>部分, 在<system.web>项组中配置应用程序的定制错误页面。下面是一个例子:

```
<configuration>
  <system.web>
    <customErrors defaultRedirect="url" mode="On|Off|RemoteOnly">
      <error statusCode="statusCode" redirect="url"/>
    </customErrors>
  </system.web>
</configuration>
```

```
</customErrors >
</system.web>
</configuration>
```

上述配置中的参数说明如下。

(1) **defaultRedirect**: 用来定义当错误发生时默认的重定向 URL。

(2) **mode**: 错误模式, 用来指定是否显示一个 ASP.NET 错误信息。Mode 的默认值为 **RemoteOnly**, 表示只把定制错误显示给不在服务器上的浏览器, 这样可让用户看到定制错误页面, 而开发人员看到标准的错误页面, 便于调试。当 Mode 设置为 **On** 时, 表示无论用户在服务器本机还是在其他地方, 都将显示定制错误页面。当 Mode 设置为 **Off** 时, 表示不被任何用户显示定制错误页面。

(3) **error** 标识符: 自定义重定向, 用来指定当出现错误时, 可以根据返回的 HTTP 状态码将用户重定向到不同的自定义错误页。在这种特定的情况下, 将忽略 **defaultRedirect** 值设定的重定向 URL, 这样可以灵活地响应不同的错误。例如, 404 Page Not Found 和 403 Access Forbidden 错误的响应可以显示不同的错误页面。

**【例 9-1】** 体现 ASP.NET 配置的应用。

设计步骤如下:

(1) 新建一个网站“Chapter9”, 在此网站中添加 2 个网页, 分别命名为“example9-1.aspx”和“error.aspx”。

(2) 切换到 example9-1.aspx 的设计视图, 添加 1 个 FileUpload 控件和 1 个 Button 控件到此页面, Button 控件的 Text 属性设置为“上传”。双击 Button 控件, 在 Button\_Click 方法体内添加如下代码:

```
string path= Request.PhysicalApplicationPath;           //获取网站的根目录的
                                                         物理路径
FileUpload1.PostedFile.SaveAs(path + FileUpload1.FileName); //上传文件
```

(3) 打开配置文件 web.config, 把<appSettings/>修改为下面代码:

```
<appSettings>
  <add key="AppKey" value="网页错误!" />
</appSettings>
```

(4) 在<system.web>项组中添加如下代码:

```
<httpRuntime executionTimeout="90" maxRequestLength="3"/> //请求最大不超过 3KB
<customErrors defaultRedirect="error.aspx" mode="On"></customErrors> //错误时重定向到
                                                         error.aspx 页
```

(5) 打开 error.aspx.cs 代码页, 在 Page\_Load 方法内添加如下代码:

```
Response.Write(ConfigurationSettings.AppSettings["AppKey"].ToString()); //在页面上输出键
                                                         AppKey 的值
```

(6) 打开 example9-1.aspx 页面, 按【Ctrl+F5】组合键运行网页, 单击【浏览】按钮选择一个大于 3KB 的文件, 单击【上传】按钮, 结果重定向到 error.aspx 页面, error.aspx 页面读取键 AppKey 的值并显示在页面上, 如图 9.1 所示。



图 9.1 运行后的结果

## 9.2 高速缓存

下面将要介绍提高 ASP.NET 应用程序性能的一种方法——高速缓存。它可以极大地提高整个应用程序的性能，从而有效地扩展系统的可伸缩性。

### 9.2.1 ASP.NET 缓存概述

当用户提出请求时，高速缓存允许把请求的结果保存到内存中。当提出后续请求时，可以直接传递高速缓存的结果，不需要重复工作。在 Web 应用程序的上下文中，高速缓存意味着在 HTTP 请求之间，内存中保持缓存的 Web 页或数据，可以重复使用它们，不必重新创建它们。对于那些可能被频繁访问的页面或数据，可以将其高速缓存下来，以提高系统的性能。

ASP.NET 提供了一种高速缓冲存储器以存储 Web 对象。实际上，为了提高系统性能，可以将高速缓冲存储器的位置指定在客户机或服务端。

#### 1. 客户端高速缓存

为改善性能，客户端程序（如浏览器）通过将 Web 上的数据保存在本地硬盘中或者用户计算机内存中来执行高速缓存的操作。但这些高速缓存不能跨多客户机共享，图 9.2 描述了客户端高速缓存的概念。

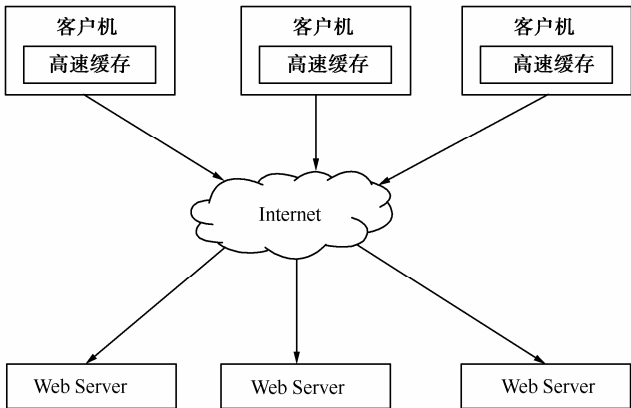


图 9.2 客户端高速缓存

#### 2. 专用服务器高速缓存

可以在服务器端执行高速缓存，这样网络上的多个用户就可以共享这些高速缓冲存储器。

我们通常使用代理服务器作为高速缓冲存储器，将频繁使用的网页存储在代理服务器的硬盘上，代理服务器不用将请求发送到实际的 Web 服务器就能满足用户的请求，这样可以提高访问速度，而多个客户机又能够共享缓存。图 9.3 描述了专用服务器高速缓存的概念。

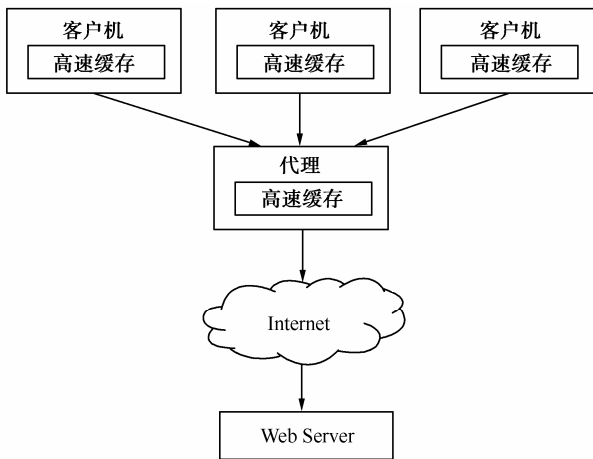


图 9.3 专用服务器高速缓存

3. 反向代理

也可以将高速缓冲存储器直接放在某个特定的 Web 服务器之前，从而减少 Web 服务器收到的请求的数量。这种模型允许代理服务器频繁响应收到的请求，并向 Web 服务器传递其他请求。这种代理的形式称为反向代理，其中 Web 服务器使用代理服务器来加速请求的处理。这种模型是唯一的，它为多个客户机高速缓存对象，而这些对象通常来自于一个服务器。图 9.4 描述了反向代理高速缓存的概念。

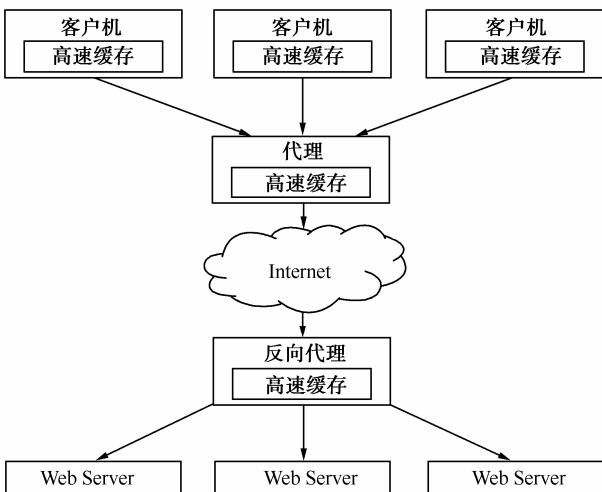


图 9.4 反向代理高速缓存

## 9.2.2 页面输出缓存

页面输出缓存是一种最简单的缓存机制，其基本原理是将用户经常访问的页面缓存到内存或硬盘中，当用户再次请求时，不需要再次执行相应的代码，而是直接把生成过的、已经缓存的 **HTML** 页面发送到客户端显示。可见，页面输出缓存是一项非常有效的增强访问性能的技术。一旦缓存被定制，那么网页就会在第一次被访问时生成缓存（Cache），直到请求过期为止。页面输出缓存的定制方法可以使用 `@OutputCache` 指令的高级技术实现。

将整个 Web 页的输出缓存，供后续请求使用。可以在页的顶部使用一条 `OutputCache` 指令来设置使用页面输出缓存，其完整语法如下：

```
<% @ OutputCache Duration="#ofseconds"
    Location="Any | Client | Downstream | Server | None | ServerAndClient "
    Shared="True | False"
    VaryByControl="controlname"
    VaryByCustom="browser | customstring"
    VaryByHeader="headers"
    VaryByParam="parametername"
    VaryByContentEncoding="encodings"
    CacheProfile="cache profile name | ""
    NoStore="true | false"
    SqlDependency="database/table name pair | CommandNotification"
%>
```

其中，常用属性如下。

(1) **Duration**：过期时间，即缓存的对象在内存中保持的秒数。

(2) **Location**：用于指定高速缓存该页面的位置。设置为 **Server** 时，只有运行该应用程序的服务器允许高速缓存该页面。**Downstream** 设置表示任何中间网络代理都可以高速缓存该页面的拷贝。**Client** 设置表示浏览器可以在本地高速缓存该页面。**Any** 设置表示上述位置均可高速缓存该页面。**None** 设置表示禁止使用任何高速缓存。

(3) **VaryByControl**：允许在服务器上高速缓存控件，使用该参数可以在当控件显示在页面上时高速缓存该控件。例如，如果有一个控件显示新项目列表，则把这个控件放在高速缓存中 10 分钟，项目就会缓存 10 分钟。

(4) **VaryByCustom**：可以指定是否为不同的浏览器保存不同的高速缓存版本，或者由指定的字符串来改变。如果设置为 **browser**，就按浏览器名称和主要版本号创建不同的高速缓存，使一个页面具有不同的高速缓存版本。当需要在不同的浏览器或设备上显示不同的输出结果时，这个参数是很有效的，它可以详细指定要高速缓存的页面的各个部分。

(5) **VaryByHeader**：根据 HTTP 头值来缓存页。

(6) **SqlDependency**：标识一组数据库/表名称对的字符串值，页或控件的输出缓存依赖于这些名称对。请注意，`SqlCacheDependency` 类监视输出缓存所依赖的数据库中的表，因此当更新表中的项时，使用基于表的轮询时将从缓存中移除这些项。如果以值 `CommandNotification` 使用通知（在 Microsoft SQL Server 2005 中），则最终会使用 `SqlDependency` 类向 SQL Server 2005 服务器注册查询通知。



(7) **VaryByParam**: 根据 GET 或 POST 请求中收到的参数来缓存页。如果该属性指定的参数发生变化, 则 ASP. NET 可以缓存不同的页。可取值如下。

- ① \* : 对 GET 或 POST 请求中的不同参数值都分别进行缓存。
- ② None : 只缓存无参数的请求。
- ③ 参数列表 : 对多个参数条件进行缓存, 如 uid、uname。

例如:

```
<%@OutputCache Duration = "20" VaryByParam = "None"%> //只缓存无参数的请求
```

**【例 9-2】** 体现页面输出缓存的应用。

设计步骤如下:

- (1) 在网站 “Chapter9” 中添加 1 个网页, 命名为 “example9-2.aspx”。
- (2) 切换到 example9-2.aspx 的源视图, 添加 OutputCache 指令来设置使用页面输出缓存, 代码如下所示 (黑体部分):

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="example9-2.aspx.cs"
    Inherits="example9_2" %>
<%@OutputCache Duration = "20" VaryByParam="None"%> //只缓存无参数的请求
```

- (3) 打开 example9-2.aspx.cs 代码页, 在 Page\_Load 方法体内输入以下代码:

```
Response.Write(DateTime.Now); //在页面上输出当前时间
```

- (4) 按 **【Ctrl+F5】** 组合键运行网页, 按 **【Ctrl+R】** 组合键刷新网页, 此时页面的时间不变, 直到 20 秒后, 运行的结果如图 9.5 所示。



图 9.5 example9-2 网页运行的结果

9.2.3 页面部分缓存

对页面进行缓存可以大大提高 Web 应用程序的性能。不过, 在有些情况下, 需要缓存页面的大部分内容, 但页面中的某些片段是动态的。例如, 创建一个页面, 如果其中的新闻故事在设定时间段内完全是静态的, 则可以设置为缓存整个页面; 如果希望提供在每次页请求时都发生变化的交替出现的广告横幅, 则该页中包含该广告的部分需要是动态的。

若允许缓存某个页面但动态地替换其中的某些内容, 则可以使用 ASP.NET 缓存后替换。通过使用缓存后替换, 将对整个页面进行输出缓存, 并将特定的部分标记为不进行缓存。

可以通过 Substitution 控件的使用实现缓存后替换。Substitution 控件指定缓存页中动态创建而不进行缓存的部分。将 Substitution 控件放置在该页上要显示动态内容的位置。

在运行时，Substitution 控件调用 MethodName 属性指定的方法。该方法必须返回一个字符串，然后该字符串替换 Substitution 控件的内容。该方法必须是 Page 或 UserControl 包含控件上的静态方法。

使用 Substitution 控件可以将客户端可缓存性更改为服务器可缓存性，以便该页面不会在客户端上进行缓存。这样可以确保以后对该页的请求能够再次调用该方法以生成动态内容。

**【例 9-3】** 利用 Substitution 体现页面部分缓存的应用。

设计步骤如下：

(1) 在网站“Chapter9”中添加 1 个网页，命名为“example9-3.aspx”。切换到设计视图，从工具箱的标准栏中拖放 1 个 Substitution 控件到页面上，Substitution 的 MethodName 属性设置为“GetUpdatedTime”。

(2) 切换到 example9-3.aspx 的源视图，添加 OutputCache 指令来设置使用页面输出缓存，代码如下所示：

```
<%@OutputCache Duration="20" VaryByParam="None"%> //只缓存无参数的请求
```

(3) 打开 example9-3.aspx.cs 代码页，在 Page\_Load 方法体内输入以下代码：

```
Response.Write(DateTime.Now.ToString()); //在页面上输出当前时间
```

(4) 并且添加 GetUpdatedTime 方法，代码如下所示：

```
public static string GetUpdatedTime(HttpContext context)
{ return DateTime.Now.ToString(); }
```

(5) 按【Ctrl+F5】组合键运行网页，按【Ctrl+R】组合键刷新网页，此时页面的上面时间不变，下面时间时刻在变化，直到 20 秒后，运行结果如图 9.6 所示。

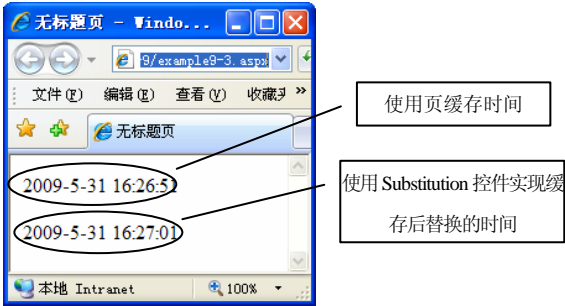


图 9.6 缓存后替换示例

Substitution 控件的 MethodName 属性值是 GetUpdatedTime，在该控件的位置显示的是 GetUpdatedTime 方法返回的当前时间，它会随着页面的刷新而变化。

### 9.2.4 应用程序数据缓存

页面输出缓存、页面部分缓存都是声明性的缓存，它们是用 OutputCache 指令来控制缓存的。ASP.NET 还包含对高速缓存对象的编程支持。

应用程序数据缓存也称为编程高速缓存，它并不缓存整个或局部页，而是允许在 HTTP

请求之间把特定的对象缓存在内存中。ASP.NET 将应用程序的“数据高速缓存”封装在 Cache 类中，它属于 System.Web.Caching 名称空间。Cache 对象的生存期与应用程序的生存期相同，当重新启动应用程序时，也会重新创建高速缓存。

Cache 类提供了强大的功能，允许自定义如何缓存项及缓存多长时间。例如，当缺乏系统内存时，缓存会自动移除很少使用的或优先级较低的项以释放内存。该技术称为清理，这是缓存确保过期数据不占用资源的方式之一。

### 1. 向应用程序缓存中添加项

向应用程序数据缓存中添加项有三种方式。

(1) 通过键和值直接设置项，向缓存中添加项。

```
Cache["myCacheItem1"] = "value1";           //将名为 myCacheItem1 的项添加到 Cache 对象中
```

(2) 使用 Insert 方法向缓存中添加项。

```
Cache.Insert("myCacheItem1", "value1");      //调用 Insert 方法向缓存中添加名为 myCacheItem1 的项
```

(3) 使用 Add 方法向缓存中添加项。

```
Cache.Add("myCacheItem1", "value1", null, DateTime.Now.AddSeconds(60), TimeSpan.Zero, CacheItemPriority.High, onRemove);
```

上面的代码是在缓存中添加一项，其键为 myCacheItem1，值为 value1，绝对过期时间为 60 秒，并具有高缓存优先级，优先级为 High。当缓存对象发生过期时，将调用 onRemove 方法。

### 2. 从应用程序缓存中删除项

ASP.NET 缓存中的数据是易失的，不能永久保存。缓存中的数据可能会被自动删除，原因有：

- (1) 缓存已满；
- (2) 该项已过期；
- (3) 依赖项发生更改。

除了自动删除外，还可以调用 Cache 类的 Remove 方法来删除缓存项。例如：

```
Cache.Remove("myCacheItem1");                //删除名为 myCacheItem1 的缓存项
```

### 3. 检索缓存项的值

要访问缓存中的数据，应指定缓存项的键名。使用过程中，由于缓存项可能已过期而被删除，因此最好先确定该项是否存在，若存在再进行删除。例如：

```
string cachedString;
if (Cache["myCacheItem1"] != null)           //判断 Cache 对象中名为 myCacheItem1 的项是
                                                否为 null
{
    cachedString = (string)Cache["myCacheItem1"];
}
else
```

```
{
    Cache.Insert("myCacheItem1", "value1"); //重新创建缓存项
    cachedString = (string)Cache["myCacheItem1"];
}
```

#### 4. 缓存依赖

可以在缓存对象和被依赖的对象（如文件、目录和数据库表等）之间建立一个有效的关联，当被依赖的对象发生变化、缓存对象变得不可用时，将自动从缓存中删除并重新创建新的缓存对象。

**【例 9-4】** 利用 DataSet 对象进行缓存，提高数据库访问的效率。

设计步骤如下：

(1) 在网站“Chapter9”中添加 1 个网页，命名为“example9-4.aspx”。切换到设计视图，从工具箱的标准栏中拖放 1 个 GridView 控件和 1 个 Lable 控件到页面上。

(2) 打开 example9-4.aspx.cs 代码页，添加命名空间：

```
using System.Data.SqlClient;
```

(3) 在 Page\_Load 方法体内输入以下代码：

```

DataView mysource;                // DataView用于创建 DataTable中所存储的数据的不同视图
mysource = (DataView)Cache["MyDataSet"];
if (mysource == null)
{
    SqlConnection MyConnection =
        new SqlConnection("DataSource=localhost;InitialCatalog=XSCJ;Integrated Security=
True");
    SqlDataAdapterMycommand = new SqlDataAdapter("select * from XS", MyConnection);
    DataSet ds = new DataSet();
    Mycommand.Fill(ds, "XS");
    mysource = new DataView(ds.Tables["XS"]);
    Cache["MyDataSet"] = mysource;
    Label1.Text = "从数据库获取数据";
}
else
{
    Label1.Text = "使用已有的缓存项";
}
GridView1.DataSource=mysource;    //设置数据源
GridView1.DataBind();             //绑定数据源

```

(4) 按【Ctrl+F5】组合键运行网页，运行结果如图 9.7 所示；按【Ctrl+R】组合键刷新网页，运行结果如图 9.8 所示。

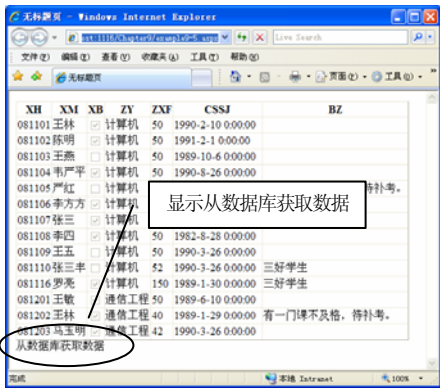


图 9.7 从数据库中获取数据

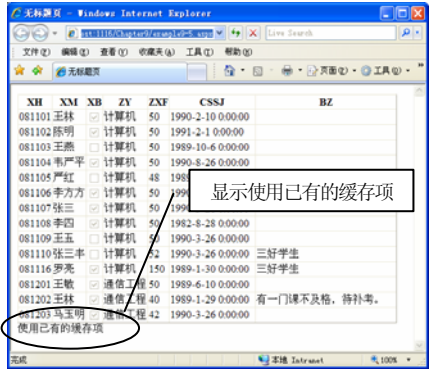


图 9.8 从缓存中获取数据

代码中通过缓存名为 `MyDataSet` 的 `DataSet` 对象实现数据缓存，其好处是当再访问该页时，可以不再从数据库中读出数据，而是直接从缓存中读出。数据缓存的性能一般比页面输出缓存的性能更好一些，原因是页面输出缓存每次仍然要从服务器端将数据取回。

## 9.3 ASP.NET XML编程

XML 是 Extensible Markup Language 的缩写，它是一种可扩展的标记语言。XML 的重要之处在于其信息处于文档之中，而显示指令在其他位置。也就是说，内容和显示是相互独立的。XML 是用于数据交换的 Web 语言，它已成为在 Internet 上传递数据的事实标准。

### 9.3.1 XML基本概念

XML 不像 HTML 那样提供了一组事先定义好的标记，而是提供了一个标准。利用这个标准，可以根据实际需要定义新的标记语言，并为这个标记语言规定它特有的一套标记。准确地说，XML 是一种元标记语言，它允许开发人员根据所提供的规则，制订各种各样的标记语言。

#### 1. XML文档的组成

HTML 提供了固定的预定义元素集，可以使用这些元素来标记一个 Web 页的各个组成部分。而 XML 没有预定义的元素，用户可以创建自己的元素，并自行命名。

**【例 9-5】** 在网站 Chapter9 的 App\_Data 文件夹中添加 XML 文件。

设计步骤如下：

打开“解决方案资源管理器”→右击“App\_Data”→选择“添加新项”→选择“XML 文件”→命名为“books.xml”→单击“添加”按钮。添加了 1 个命名为 books.xml 的文件，在此文件中添加 XML 代码，如下所示：

```
<?xml version="1.0" encoding="utf-8" ?> //此行代码是系统添加的
<!-- This file is a part of a book store inventory database --> //在 XML 文档中的注释
<inventory>
  <book>
    <title>ASP.NET(C#)3.5</title>
    <author>郑阿奇</author>
```

```
<pages>298</pages>
<price>25.60</price>
</book>
<book >
  <title >C++程序设计</title>
  <author >丁有和</author>
  <pages>598</pages>
  <price>45.60</price>
</book>
</inventory >
```

一个 XML 文档主要由两个部分组成：序言和文档元素（也就是根元素）。在文档元素之后可以包括注释、处理指令和空格。

### （1）序言

第一行是 XML 声明。它说明这是一个 XML 文档，并且给出版本号。

### （2）文档元素

XML 文档中的元素是以树形分层结构排列的，元素可以嵌套在其他元素中。文档必须只有一个顶层元素，称为文档元素或根元素，类似于 HTML 页中的 BODY 元素，其他所有元素都嵌套在其中。在【例 9-5】中，文档元素是 inventory，其起始标签是<inventory>，结束标签是</inventory>，内容是 2 个嵌套的 book 元素。

在 XML 文档中，元素指出了文档的逻辑结构，并且包含了文档的信息内容。一个典型的元素有起始标签、元素内容和结束标签。元素内容可以是字符、数据、其他（嵌套的）元素或两者的组合。

## 2. 创建XML文档的基本规则

创建格式正确的 XML 文档的一些基本规则如下。

（1）文档必须有一个顶层元素（文档元素或根元素）。所有其他元素必须嵌入到其中。

（2）元素必须被正确地嵌套。也就是说，如果一个元素在另一个元素中开始，那么它必须在同一个元素中结束。

（3）每一个元素必须同时拥有起始标签和结束标签。与 HTML 不同，XML 不允许忽略结束标签。

（4）起始标签中的元素类型名必须与相应结束标签中的名称完全匹配。

（5）元素类型名是区分大小写的。实际上，XML 标记中的所有文本都是区分大小写的。例如，下列元素是非法的，因为起始标签的类型名与结束标签的类型名不匹配：

```
<TITLE>Leaves of Grass</title>
```

//非法的 XML 元素

## 3. 给元素添加属性

在一个元素的起始标签中，可以包含一个或多个属性。属性由属性名、等号及属性值组成。属性名可以由用户任意定义。例如，下面的 PRICE 元素包含一个名为 Type 的属性，它被赋值为 retail。

```
<price Type= "retail" > 32.50 </price>
```

4. XML的显示

单独用 XML 是不能在页面中正确显示其数据的，必须使用某种格式化技术，如 CSS 或 XSL，才能显示 XML 标记创建的文档。

XML 是将数据和格式分离的，所以 XML 文档本身不知道如何来显示数据，必须有辅助文件来帮助实现。辅助 XML 来设定显示风格样式的文件类型有如下两种。

(1) XSL

XSL 全称为 Extensible Stylesheet Language（可扩展样式语言），是用来设计 XML 文档显示样式的主要文件类型。XSL 也可以将 XML 转化为 HTML。

(2) CSS

CSS 全称为 Cascading Style Sheets（层叠样式表），是目前用来在浏览器上显示 XML 文档的主要方法。

9.3.2 XML数据访问

.NET 框架为读写 XML 数据提供了两个核心类，即 XmlReader 类和 XmlWriter 类，它们属于 System.Xml 命名空间。

1. 用XmlReader读取XML数据

XmlReader 能够从流或者 XML 文档中读取 XML 数据。该类提供了对 XML 数据快速、非缓存、只读和只进的访问方式。XmlReader 提供的方法和属性可以浏览 XML 数据并读取节点的内容。

使用 XmlReader 类访问 XML 数据的步骤如下。

(1) 使用 XmlReader 类的 Create()方法创建该类的一个实例，并将 XML 文件名作为参数传递给 Create()方法。

(2) 循环调用 Read()方法。该方法从文件的第一个节点开始，一次调用只读取一个节点。如果存在一个节点可被读取则返回 True，而当到达文件尾部时，则返回 False。

(3) 在这个循环中，将检查 XmlReader 对象的属性和方法，以获得关于当前节点的信息（节点类型、名称和数据等）。不断地执行循环直到 Read()返回 False 为止。XmlReader 类具有大量的属性和方法，表 9.1 列出了其主要属性和方法。

表 9.1 XmlReader 类的主要属性和方法

属性/方法	说 明
HasAttributes	判断当前节点是否有任何属性
HasValue	判断当前节点是否可以具有 Value
IsEmptyElement	判断当前节点是否为空元素（如<MyElement/>）
Name	获取当前节点的限定名
NodeType	以 XmlNodeType 枚举的形式返回当前节点的类型
ReadState	以 ReadState 枚举的形式返回读取器的状态
Value	获取当前节点的文本值
Close()	将 ReadState 更改为 Closed，以关闭 XmlReader 对象

属性/方法	说 明
Create()	创建一个新的 XmlReader 实例，并将其返回给调用程序
GetAttribute()	获取属性的值
IsStartElement()	测试当前内容节点是否是开始标记
MoveToAttribute()	将读取器移动到指定的属性
MoveToNextAttribute()	移动到下一个属性
Read()	从流中读取下一个节点
ReadEndElement()	检查当前内容节点是否为结束标记并将读取器推进到下一个节点

在使用 XmlReader 实现读取 XML 数据时，通常会使用 XmlReaderSettings 类的实例与 XmlReader 配合来完成读取任务。Create 方法是获取 XmlReader 实例的首选机制。而 Create 方法使用 XmlReaderSettings 类指定要在创建的 XmlReader 对象中实现哪些功能。

**【例 9-6】** 使用对 XmlReader 和 XmlReaderSettings 读取【例 9-5】所创建的 book.xml 文件。设计步骤如下：

- (1) 在网站“Chapter9”中添加 1 个网页，命名为“example9-6.aspx”。
- (2) 打开 example9-6.aspx.cs 代码页，添加命名空间：

```
using System.Xml;
```

(3) 在 Page\_Load 方法体内输入以下代码：

```
int bookcount = 0; //书籍的计数器
string s = null; //检索的书名存放在 s 中
XmlReaderSettings settings = new XmlReaderSettings();
settings.IgnoreWhitespace = true; //忽略空白内容
settings.IgnoreComments = true; //忽略注释内容
string booksFile = Server.MapPath("~/App_Data/books.xml");
XmlReader reader = XmlReader.Create(booksFile, settings);
while (reader.Read())
{
    //如果节点类型为 Element，且本地名为 title，则计数器加 1，书名存放在 s 中
    if (reader.NodeType == XmlNodeType.Element && reader.LocalName == "title")
    {
        bookcount++;
        s+=reader.ReadElementString("title")+"<br>";
    }
}
reader.Close();
Response.Write(String.Format("共找到{0}本书:<br>", bookcount));
Response.Write(s);
```



(4) 按【Ctrl+F5】组合键运行网页，运行结果如图 9.9 所示。

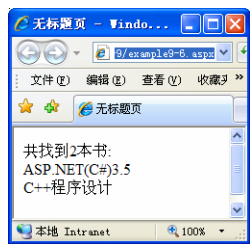


图 9.9 从 book.xml 中检索出的书本信息

注意：XmlReaderSettings 的 IgnoreWhitespace 和 IgnoreComments 被设置为 true，表示忽略文档的空白内容和注释内容。

2. 用XmlWriter写入XML数据

.NET Framework 提供了一个编写器 XmlWriter，该编写器提供一种快速、非缓存和只进的方式来生成包含 XML 数据的流或文件。

XmlWriter 类具有大量的属性和方法，表 9.2 列出了其主要属性和方法。

表 9.2 XmlWriter 类的主要属性和方法

属性/方法	说 明
Settings	获取用于创建此 XmlWriter 实例的 XmlWriterSettings 对象
WriteState	获取编写器的状态
Close()	关闭当前流和基础流
Create()	创建一个新的 XmlWriter 实例
WriteAttributeString()	写出具有指定值的属性
WriteElementString()	写出包含指定字符串值的元素
WriteEndAttribute()	关闭上一个 WriteStartAttribute 调用
WriteEndDocument()	关闭任何打开的元素或属性并将编写器重新设置为 Start 状态
WriteStartAttribute()	编写属性的起始内容
WriteStartDocument()	编写 XML 声明
WriteStartElement()	写出指定的开始标记
WriteString()	编写给定的文本内容
WriteValue()	编写单一的简单类型化值

XmlWriter 类还有一个设置类 XmlWriterSettings，用于指定要在新的 XmlWriter 对象上启用的功能集。XmlWriterSettings 类的属性可以启用或禁用功能，通过将 XmlWriterSettings 对象传递给 XmlWriter 类的 Create 方法，指定要支持的编写器功能。

【例 9-7】 使用对 XmlWriter 和 XmlWriterSettings 创建一个 XML 文档并输出到页面上。设计步骤如下：

(1) 在网站“Chapter9”中添加 1 个网页，命名为“example9-7.aspx”。切换到源视图，

删除下面代码的所有代码。

```
<% @PageLanguage="C#" AutoEventWireup="true" CodeFile="example9-7.aspx.cs" Inherits="example9_7" %>
```

(2) 打开 example9-7.aspx.cs 代码页，添加命名空间：

```
using System.Xml;
```

(3) 在 Page\_Load 方法体内输入以下代码：

```
Double price = 36.00;
DateTime publicationdate = new DateTime(2009,6,1);
String author = "郑阿奇";
XmlWriterSettings settings = new XmlWriterSettings();
settings.Indent = true; //设置缩进
settings.NewLineOnAttributes = false; //属性在一行中显示
Response.ContentType = "text/xml"; //设置输出流的类型
XmlWriter writer = XmlWriter.Create(Response.OutputStream, settings);
writer.WriteStartDocument(); //编写 XML 声明
writer.WriteStartElement("inventory"); //创建开始标记<inventory>
writer.WriteStartElement("book"); //创建开始标记<book>
writer.WriteStartAttribute("出版日期"); //添加属性“出版时间”
writer.WriteValue(publicationdate); //给属性赋值
writer.WriteEndAttribute(); //关闭上一个 WriteStartAttribute 调用
writer.WriteElementString("标题", "ASP.NET3.5"); //写出包含指定字符串值的元素
writer.WriteStartElement("价格"); //写出指定的开始标记
writer.WriteValue(price);
writer.WriteEndElement(); //关闭创建<价格>元素
writer.WriteStartElement("作者"); //写出开始标记<作者>
writer.WriteValue(author);
writer.WriteEndElement(); //关闭创建<作者>元素
writer.WriteEndElement(); //关闭创建<book>元素
writer.WriteEndElement(); //关闭创建<inventory>元素
writer.WriteEndDocument();
writer.Close();
```

(4) 按【Ctrl+F5】组合键运行网页，运行结果如图 9.10 所示。

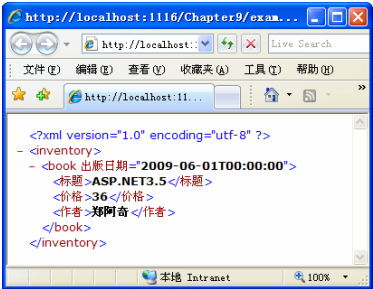


图 9.10 输出所创建的 XML 文档

## 习 题

1. ASP.NET 为开发者提供了一个基于\_\_\_\_\_格式的配置文件 Web.config。
2. Web.config 中所有配置信息都在\_\_\_\_\_根元素之间。
3. 控件缓存和缓存后替换有什么区别？
4. 如何访问 XML 文档中的数据，XmlReader 和 XmlWriter 的作用是什么？

## 第 10 章 Web服务

以前, 创建能够在网络尤其是在 Internet 上远程调用的组件一直是一件困难的事情。ASP.NET Web Services 模型提供了一种简单而直接的方式, 使 Web 开发人员可以利用它, 通过 Internet 提供的标准通信方式, 共享应用逻辑。

### 10.1 Web服务的基本概念

要完全理解 Web 服务的影响, 需要了解分布式计算方面的知识。

#### 10.1.1 基于组件的分布式计算概念

什么是分布式计算? 分布式计算是将应用程序逻辑分布到网络上的多台计算机中。之所以要把应用程序逻辑进行分布的原因有多种:

- (1) 分布式计算使得链接不同的机构或团体成为可能;
- (2) 应用程序访问的数据通常位于不同的计算机上, 应用程序逻辑应当靠近数据所在的计算机;
- (3) 分布式应用程序逻辑可以在多个应用程序间重用, 升级分布式应用程序块时不必升级整个应用程序;
- (4) 通过分布应用程序逻辑, 使得负载分摊到不同的计算机上, 从而提供了潜在的性能优化;
- (5) 当新的需要产生时, 应用程序逻辑可以重新分布或者重新连接;
- (6) 扩展一层比扩展整个应用程序容易。

随着 Internet 的不断发展, Internet 增强了分布式计算的重要性和适用性。Internet 的简单易用和无处不在的特性使得分布式计算作为分布式应用的骨干成为必然的选择。

#### 10.1.2 什么是Web服务

可以从技术上给 Web 服务下一个定义: Web 服务是以独立于平台的方式, 通过标准的 Web 协议, 可以由程序访问的应用程序逻辑单元。

对该定义说明如下。

- (1) 应用程序逻辑单元: Web 服务包括一些应用程序逻辑单元或者代码。这些代码可以完成运算任务, 可以完成数据库查询, 可以完成计算机程序能够完成的任何工作。
- (2) 可由程序访问: 当前大多数 Web 站点都是通过浏览器由人工访问的, Web 服务可以由计算机程序来访问。
- (3) 标准的 Web 协议: Web 服务的所有协议都基于一组标准的 Web 协议, 如 HTTP、XML、SOAP、WSDL、UDDI 等。
- (4) 平台独立性: Web 服务可以在任何平台上实现。因为标准协议不是由单个供应商专用的, 它由大多数主要供应商支持。

Web 服务允许分布式应用程序通过网络（通常是 Internet）共享业务逻辑。例如，证券公司提供股票报价服务，咨询机构使用其报价服务。

Web 服务使用可以超越各种机器平台和操作系统的通用协议（HTTP/HTTPS）和通用语言（XML），因此它非常适合在 Internet 上实现业务逻辑的共享服务。图 10.1 演示了客户机调用 Web 服务的方法时的工作流程。

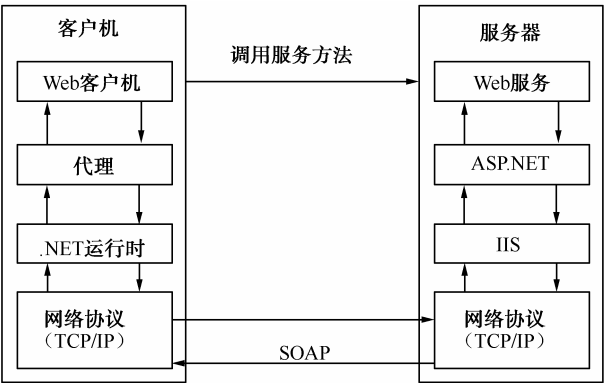


图 10.1 Web 服务的工作流程

## 10.2 ASP.NET Web服务的创建与测试

建立 Web 服务就是把一些信息或逻辑对其他计算机和客户公开。

### 10.2.1 创建Web服务

VS 2008 提供了创建 Web 服务的模板。下面通过一个实例来介绍如何创建 Web 服务。

**【例 10-1】** 创建简单的 Web 服务。

设计步骤如下：

（1）运行 VS 2008→单击菜单中的“文件”→“新建”→“网站”，在弹出的“新建网站”对话框中选择“ASP.NET Web 服务”模板，如图 10.2 所示，然后单击【确定】按钮。

（2）打开“解决方案资源管理器”，其中包含一个 Web 服务 Service.asmx，其代码隐藏文件 Service.cs 位于 App\_Code 文件夹中。需要注意的是，Web 服务文件的扩展名为“asmx”，如图 10.3 所示。

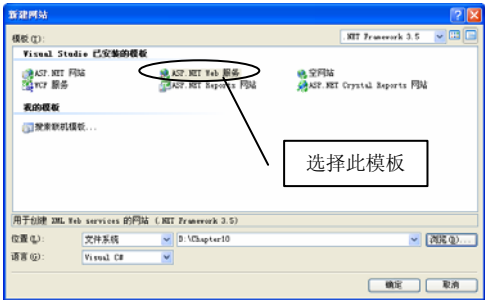


图 10.2 创建 Web 服务

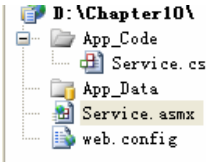


图 10.3 所创建的 Web 服务文件

## 10.2.2 @WebService指令

打开【例 10-1】的 Web 服务文件 Service.asmx，该文件只包含 WebService 页面指令，如下：

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs" Class="Service" %>
```

从上面的代码可以看出，Web 服务文件使用 @WebService 指令代替了普通网页的 @Page 指令。简单的 @WebService 指令只有 4 个属性。

(1) **Class**：必选属性，指定用于定义方法和数据类的类。这些方法和数据类对于使用 Web 服务的客户机来说是可见的。

(2) **CodeBehind**：如果采用代码隐藏模型，则该属性是必需的。该属性指定了实现 Web 服务的源文件，默认情况下 Web 服务的代码隐藏文件放在 App\_Code 文件夹中。

(3) **Language**：必选属性，指定用于 Web 服务的编程语言。

(4) **Debug**：指示是否应使用调试符号编译 XML Web services。如果应使用调试符号编译 XML Web services，则为 true；否则为 false。

## 10.2.3 Web服务类

打开【例 10-1】的 Web 服务文件 Service.asmx 的代码隐藏文件 Service.cs，代码中定义了 Web 服务类“Service”，然后，在 Service 类中包含了一个空的构造函数和一个公共方法 HelloWorld()。代码如下所示：

```
using System;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// 若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消对下行的注释。
// [System.Web.Script.Services.ScriptService]
public class Service : System.Web.Services.WebService
{
    public Service ()                //空的构造函数
    {
        //如果使用设计的组件，请取消注释以下行
        //InitializeComponent();
    }
    [WebMethod]
    public string HelloWorld()        //公共方法，返回“Hello World”
    {
        return "Hello World";
    }
}
```

### 10.2.4 WebService特性

所有的 Web 服务都封装在一个类中。这个类在类声明前通过 WebService 特性定义为 Web 服务。例如：

```
[WebService(Namespace = "http://tempuri.org/")]
```

WebService 特性有几个属性。在 Web 服务中，默认使用 WebService 特性的 Namespace 属性，其初始值为 http://tempuri.org/。这是一个临时的命名空间，用来设置 Web 服务的命名空间，强烈建议修改此命名空间，以区别于其他 Web 服务。

WebService 特性的其他属性有 Name 和 Description。Name 允许改变 Web 服务名通过 ASP.NET 测试页面显示给开发人员的方式。Description 可以提供 Web 服务的文本描述，该描述也将显示在 ASP.NET 测试页面上。如果 WebService 特性包含多个属性，则可用逗号分隔，示例如下：

```
[WebService(Namespace = "http://tempuri.org/",Name="HelloWorld")]
```

### 10.2.5 定义Web服务方法

可通过 Web 进行通信的 Web 服务的方法称为 Web 服务方法。实现 Web 服务类的方法不会自动拥有通过 Web 进行通信的功能，但是对于使用 ASP.NET 创建的 Web 服务，添加该功能则十分简单，只需在公共方法前加上 WebMethod。

Web 服务类可以包含任意多个方法，可以是标准方法，也可以是 Web 方法。只有添加了 WebMethod 属性的方法才能通过 HTTP 访问。

**【例 10-2】** 修改【例 10-1】所创建的 Web 服务中的方法。

设计步骤如下：

- (1) 运行 VS 2008→打开“解决方案资源管理器”→打开“Service.cs”文件。
- (2) 修改 HelloWorld()代码，增加输入参数，代码如下：

```
[WebMethod]
public string HelloWorld(string userName) {    return "你好: " + userName + " ! ";    }
```

这样在调用 Web 服务方法 HelloWorld()时，需要从浏览器中输入一个用户名字字符串，然后可以返回一个字符串。

### 10.2.6 测试Web服务

**【例 10-3】** 测试【例 10-1】所创建的 Web 服务。

设计步骤如下：

- (1) 运行 VS 2008→打开“解决方案资源管理器”→打开“Service.cs”文件或者“Service.asmx”文件。
- (2) 按【Ctrl+F5】组合键运行服务，结果如图 10.4 所示。单击页面上的“HelloWorld”进入方法测试页面，如图 10.5 所示。



图 10.4 Web 服务测试页面

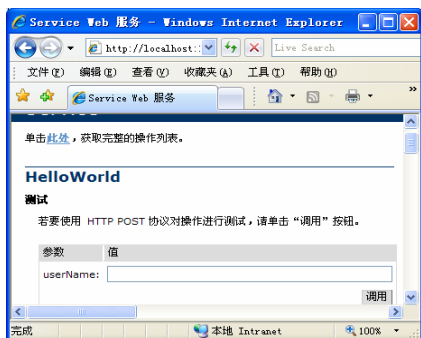


图 10.5 方法测试页面

(3) 输入参数“username”的值，这里输入“郑阿奇”，单击【调用】按钮，结果如图 10.6 所示。



图 10.6 HelloWorld 方法测试结果

页面顶部显示了 Web 服务的名称“Service”。默认情况下，Web 服务名使用类名。测试页面中，还列出了 Web 服务公开的方法名，本例中仅一个 HelloWorld 方法。

在测试页面中还显示了该 Web 服务的 Web 服务描述语言（WSDL）的文档链接“服务说明”。WSDL 文件是 Web 服务的实际接口，通过它 VS 2008 知道需要什么数据才能使用 Web 服务。WSDL 文档描述了发出请求需要执行的操作及从响应中会返回什么内容。

从图 10.6 可以看出，HelloWorld 方法的执行结果以 XML 文档的形式呈现。注意，调用该 Web 服务的 URL 是 http://localhost:1842/Chapter10/Service.asmx/HelloWorld，也就是相应的.asmx 文件，再加上 Web 服务的方法名。

## 10.3 使用ASP.NET Web服务

到目前为止已经了解了 Web 服务的创建和测试过程，下面将介绍如何在 ASP.NET 中使用 Web 服务。

### 10.3.1 添加Web引用

要在 ASP.NET 中使用 Web 服务，应创建一个 ASP.NET Web 窗体来调用 Web 服务公开的方法。下面介绍如何在 ASP.NET 中使用前面创建的 Web 服务“Service”。

**【例 10-4】** 引用【例 10-2】修改过的 Web 服务“Service”。

(1) 在【例 10-1】建立的网站中（实际上可以在新建的网站中），右击网站根目录，选择“添加 Web 引用”项。



(2) 在如图 10.7 所示的“添加 Web 引用”对话框中，单击“此解决方案中的 Web 服务”→单击“Service”，如图 10.8 所示，单击【添加引用】按钮，系统自动在 URL 列表框中填入 Web 服务的 URL，如本例中的 http://localhost:1842/Chapter10/Service.asmx/HelloWorld。然后在“添加 Web 引用”文本框中填入希望在应用程序中使用的添加 Web 引用名，本例使用默认的“localhost”。

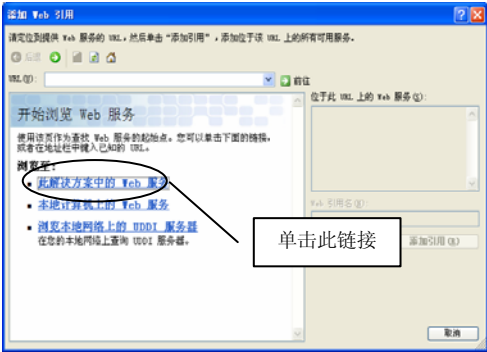


图 10.7 “添加 Web 引用”对话框

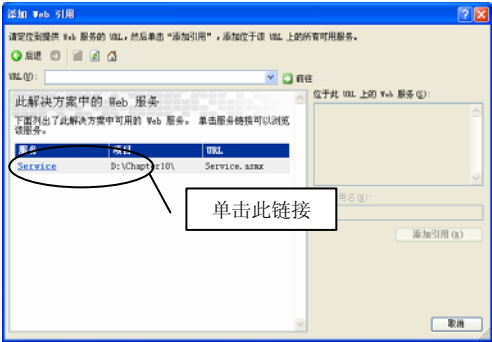


图 10.8 “添加 Web 引用”对话框

(3) 打开解决方案资源管理器，系统添加了“localhost” Web 引用，如图 10.9 所示。该引用位于 App\_WebReferences 目录中，在该目录下已经添加了该 Web 服务的 WSDL 文件。而在应用程序的 web.config 文件的<appSettings>块中，添加了如下配置，以设定实际的 Web 引用。

```
<appSettings>
    <add key="localhost.Service" value="http://localhost:1841/Chapter10/Service.asmx"/>
</appSettings>
```

localhost 引用和 Web 服务名放在一起，提供了一个键值 localhost.Service，value 属性的值是 Web 服务的位置，它位于 Service.asmx 页面。

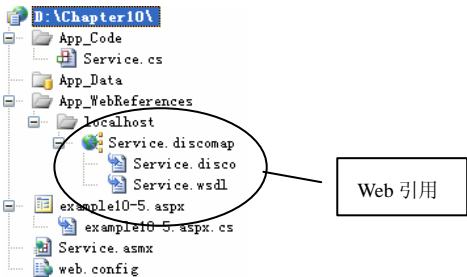


图 10.9 添加 Web 引用后的解决方案

10.3.2 客户端调用Web服务

现在，Web 服务已经添加到 ASP.NET 应用程序中，可以在 ASP.NET Web 窗体中使用 Web 服务了。

【例 10-5】 客户端调用【例 10-4】中的 Web 服务。

(1) 在【例 10-1】建立的网站中,添加网页 example10-5.aspx,在页面中添加一个 TextBox 控件、一个 Button 控件和一个 Label 控件,其中 Button 控件的 Text 属性设置为“调用 Service 服务”,如图 10.10 所示。

(2) 双击 Button 控件,添加按钮“调用 Service 服务”的单击事件代码,如下所示:

```
protected void Button1_Click(object sender, EventArgs e)
{
    localhost.Service ms = new localhost.Service();    //实例化 Service 对象
    //调用 Service 对象 ms 的 HelloWorld 方法,将 TextBox.Text 作为参数传递给 HelloWorld 方法
    Label1.Text = ms.HelloWorld(TextBox1.Text);
}
```

(3) 按【Ctrl+F5】组合键运行网页 example10-5.aspx,在浏览器窗口中输入“郑阿奇”,然后单击【调用 Service 服务】按钮,将会调用 Web 服务 Service 的 HelloWorld 方法,并将结果显示在 Label 上,效果如图 10.11 所示。



图 10.10 页面设计

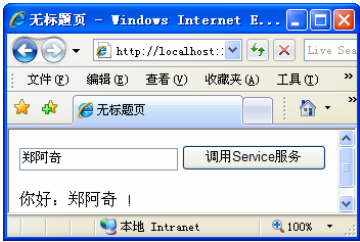


图 10.11 调用 Web 方法的运行结果

**程序分析：**在按钮“调用 Service 服务”的单击事件代码中,首先创建 Web 服务 Service 的代理 ms,然后再由 localhost.Service 代理对象的实例来调用 Web 服务的方法。

## 习 题

1. 什么是 Web 服务?
2. 如何创建 ASP.NET Web 服务?
3. 在页面中如何访问 ASP.NET Web 服务?

# 第 11 章 ASP.NET AJAX

AJAX（Asynchronous JavaScript And XML）是一种实现异步网络应用的技术，是目前 Web 应用程序中使用广泛的专门术语，在 Web 应用程序开发中，AJAX 表示建立利用 XMLHttpRequest 对象的应用程序的能力。在 JavaScript 中可以建立和包含 XMLHttpRequest 对象。另外，大多数浏览器都支持这个对象的使用，于是诞生了 AJAX 模型。

## 11.1 ASP.NET AJAX概述

微软发布的第一个 AJAX 工具集是 Atlas，后来改名为 ASP.NET AJAX，它极大地简化了在应用程序中使用 AJAX 特性的过程，使 Web 应用程序比以前更流畅。

### 11.1.1 为什么使用AJAX

AJAX 对 Web 应用程序有什么作用？首先分析一下没有使用 AJAX 的 Web 页面请求与响应的过程。图 11.1 是典型网页的请求和响应过程。

从图 11.1 可以看出，用户从浏览器向服务器上的应用程序发出请求，服务器处理请求，ASP.NET 生成一个页面，然后将该页面作为响应发送给请求者。用户接收到响应后，这个响应显示在浏览器上。

每一次请求与响应，都会是一次完整的请求与响应过程，即使用户只是想获得部分页面的更新，也将是整个页面的再一次请求与响应。这将消耗大量的网络带宽和服务器资源。

而支持 AJAX 的页面会是什么情况呢？支持 AJAX 的 Web 页面在客户机上包含一个 JavaScript 库，在请求发送时，JavaScript 库会调用 Web 服务器，得到部分页面的一个响应；接着客户机上的 JavaScript 库更新了该部分的页面，但并没有更新整个页面，只处理部分页面，于是用户就会觉得页面比较“流畅”，响应更快了。图 11.2 显示了支持 AJAX 的网页的请求与响应过程。

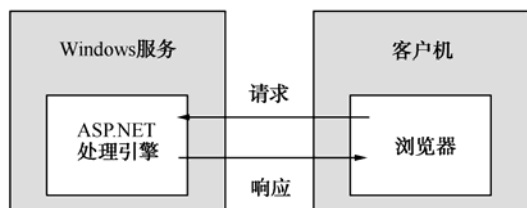


图 11.1 典型网页的请求与响应过程

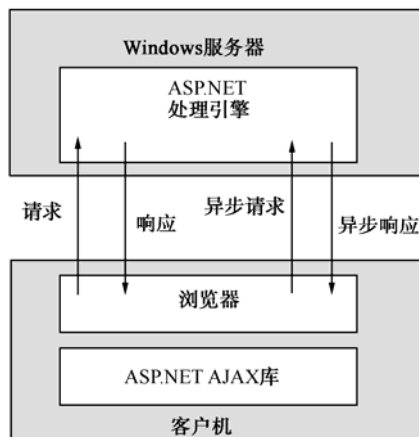


图 11.2 支持 AJAX 的网页的请求与响应过程

## 11.1.2 VS 2008 与ASP.NET AJAX

在 VS 2008 之前, ASP.NET AJAX 产品是一个独立安装的产品, 必须安装在客户机和 Web 服务器上。现在, ASP.NET AJAX 不仅是 VS 2008 的一部分, 而且还内置到 .NET Framework 3.5 中。如果用户使用的是 ASP.NET 3.5, 则使用 ASP.NET AJAX 不需要任何额外的安装。

由于 ASP.NET AJAX 是 ASP.NET 框架的一部分, 所以在创建新的 Web 应用程序时, 不需要创建特殊类型的 ASP.NET 应用程序, 使用 VS 2008 创建的所有类型的 Web 应用程序都支持 AJAX。另外, 在 .NET Framework 3.5 上建立的支持 AJAX 的应用程序, 可以工作在所有主流的浏览器上 (如 Firefox、Opera 等)。

## 11.1.3 ASP.NET AJAX客户端技术

ASP.NET AJAX 包含 2 个部分, 一部分是客户机框架和一系列完全位于客户端的服务; 另一部分就是服务器端框架。ASP.NET AJAX 的客户端主要提供客户机与服务器进行异步请求通信。因此, 微软提供了一个客户端脚本库, 它是一个 JavaScript 库, 负责必要的通信。

客户端脚本库提供了一个面向对象的 JavaScript 界面, 它与 .NET Framework 的各个方面都很一致。由于浏览器兼容组件是内置的, 所以在这一层上完成的所有工作, 或者在大多数情况下让 ASP.NET AJAX 执行的工作, 都可以在许多不同的浏览器上完成。

## 11.1.4 ASP.NET AJAX服务器端技术

ASP.NET 开发人员很可能在 ASP.NET AJAX 的服务器端方面花很多时间, 而 ASP.NET AJAX 主要是客户端技术与服务器端技术的通信。可以在 ASP.NET AJAX 的服务器端执行许多任务, 服务器端框架知道如何处理客户请求 (如输出具有正确格式的响应), 还负责在 JavaScript 对象和服务器端代码中使用的 .NET 对象之间编组对象。

# 11.2 建立ASP.NET AJAX应用程序

有两种类型的 Web 开发人员, 一类喜欢使用服务器端控件和在服务器端处理这些控件, 另一类喜欢在客户端使用 DHTML、JavaScript 处理和控制页面及其行为。

ASP.NET AJAX 是为这两类人设计的。如果更多地工作在 ASP.NET AJAX 的服务器端, 则可以使用新的 ScriptManager 控件和 UpdatePanel 控件为 ASP.NET 应用程序提供 AJAX 支持, 而编写的代码量却很少。所有这些工作都可以使用熟悉的 ASP.NET 编程模型。不过, 也可以直接使用客户端脚本库, 对客户机上发生的事件进行更多的控制。

在页面中添加 AJAX 功能, 主要方法是在页面中添加 ScriptManager 和 UpdatePanel 服务器控件, 从而使页面支持 AJAX 功能。

**【例 11-1】** 新建一个页面用来体现 AJAX 的功能。

(1) 运行 VS 2008, 新建一个网站 “Chapter11”, 在此网站上添加一个网页, 命名为 “example11-1.aspx”。

(2) 切换到设计视图，从工具箱中的 AJAX Extensions 组中拖放一个 ScriptManager 控件到页面上。再放一个 Label 控件，ID 为“Label1”，并在前面输入“服务器时间：”提示字样，如图 11.3 所示。

(3) 从工具箱的 AJAX Extensions 组中拖放一个 UpdatePanel 控件到页面上，在 UpdatePanel 控件中放入一个 ID 为“Lable2”的 Lable 控件和一个 Button 控件。在 Lable2 的前面输入“利用 AJAX 获取服务器时间：”提示字样，Button 的 Text 属性设置为“获取”。双击 Button 控件，添加 Click 事件代码，如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{ Label2.Text = DateTime.Now.ToString(); }
```

(4) 在 Page\_Load 方法体内添加代码，如下所示：

```
Label1.Text = DateTime.Now.ToString();
```

(5) 按【Ctrl+F5】组合键运行网页，单击【获取】按钮，AJAX 获取的时间在改变，如图 11.4 所示。

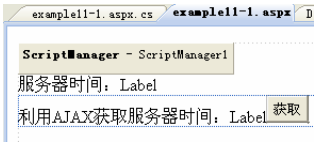


图 11.3 页面设计



图 11.4 网页运行的结果

程序分析：

运行网站时，服务器回送完整的页面，更新 Label1 控件上的当前时间。单击【获取】按钮会进行异步回送（不是完整页面回送），更新包含在 UpdatePanel 服务器控件中的内容，即 Label2 控件上的当前服务器时间，而 Label1 控件上的时间不变。

11.3 ASP.NET AJAX服务器端控件

VS 2008 工具箱的 AJAX Extensions 组中，包含一组服务器控件，它们的主要作用是 ASP.NET 应用程序添加 AJAX 支持。表 11.1 列出了 ASP.NET 3.5 中新的 ASP.NET AJAX 服务器控件。

表 11.1 ASP.NET AJAX 服务器控件

控 件	说 明
ScriptManager	管理信息的编组，为需要部分更新的页面提供支持 AJAX 的服务器。每个 ASP.NET 页面都需要一个 ScriptManager 控件来工作且只能有一个
ScriptManagerProxy	用做内容页面的 ScriptManager 控件，ScriptManagerProxy 控件位于内容页面（或子页面）上，与位于 master 页面上的 ScriptManager 控件一起工作
Timer	在指定的时间间隔执行回发，如果将 Timer 控件用于 UpdatePanel 控件，则可以按定义的时间

	间隔启用部分页面更新。也可以使用 Timer 控件来发送整个页面
--	----------------------------------

续表

控 件	说 明
UpdatePanel	允许定义页面的某些区域支持使用 ScriptManager，之后这些区域就可以回送部分页面，在正常的 ASP.NET 页面回送过程之外更新它们自己
UpdateProgress	这个控件允许给终端用户显示一个可视化元素，说明部分页面回送操作正在更新页面的某些部分。这是长时间运行 AJAX 更新时显示进度的理想控件

11.3.1 ScriptManager控件

在 ASP.NET AJAX 领域中，最重要的控件是 ScriptManager，它管理支持 AJAX 的 ASP.NET 网页的客户端脚本。默认情况下，ScriptManager 控件会向页面注册 Microsoft AJAX Library 的脚本，它负责在客户机和服务器之间来回编组信息。这将使客户端脚本能够使用类型系统扩展并支持部分页呈现和 Web 服务调用这样的功能。

每个要使用 ASP.NET 3.5 提供的 AJAX 功能的页面都需要使用一个 ScriptManager 控件。只要在页上使用 ScriptManager 控件，就启用了下列 ASP.NET 的 AJAX 功能。

- (1) Microsoft AJAX Library 的客户端脚本功能和要发送到浏览器的任何自定义脚本。
- (2) 部分页呈现，允许单独刷新页面上的部分区域而无须回发。ASP.NET UpdatePanel、UpdateProgress 和 Timer 控件需要 ScriptManager 控件才能支持部分页呈现。
- (3) Web 服务的 JavaScript 代理类，允许使用客户端脚本来访问 Web 服务和 ASP.NET 页中特别标记的方法。它通过将 Web 服务和页方法作为强类型对象公开来达到此目的。
- (4) JavaScript 类，用于访问 ASP.NET 身份验证、配置文件和角色应用程序服务。

当页包含一个或多个 UpdatePanel 控件时，ScriptManager 控件将管理浏览器中的部分页呈现。该控件与页生命周期进行交互，以更新位于 UpdatePanel 控件内的部分页。ScriptManager 控件的 EnablePartialRendering 属性确定某个页是否参与部分页更新。默认情况下，EnablePartialRendering 属性为 true。

11.3.2 ScriptManagerProxy控件

目前，许多大型的 ASP.NET 应用程序都使用母版页技术来构建模板化的 Web 站点。若母版页中添加了 ScriptManager 控件，那么所有使用母版页创建的内容页都支持 ASP.NET AJAX 功能。如果要在某个内容页中将额外的脚本和服务添加到 ScriptManager 控件所定义的脚本和服务集合中，则此时就必须使用 ScriptManagerProxy 控件（由于一个网页只能包含一个 ScriptManager 控件，所以不能在内容页中同时使用 ScriptManager 控件）。ScriptManagerProxy 控件可在母版页或宿主页已包含 ScriptManager 控件的情况下，将新增的脚本和服务添加到内容页和用户控件中。

注意：如果在内容页中使用 ScriptManagerProxy 控件，而在母版页中没有 ScriptManager 控件，就会出错。

### 11.3.3 UpdatePanel控件

UpdatePanel 控件是 AJAX 特有的控件，它是 ASP.NET 3.5 新增的控件，在处理 AJAX 时使用得最多。使用 UpdatePanel 控件可生成功能丰富的、以客户端为中心的 Web 应用程序。通过使用 UpdatePanel 控件，可以刷新页的选定部分，而不是使用回发刷新整个页面。这称为执行“部分页更新”。包含一个 ScriptManager 控件和一个或多个 UpdatePanel 控件的 ASP.NET 网页可自动参与部分页更新。当使用 UpdatePanel 控件时，页行为是独立于浏览器的，并且有可能会减少在客户端和服务器之间传输的数据量。

UpdatePanel 控件是一个容器控件，它没有相关的 UI 项，它是引发部分页面回送的方式，仅更新 UpdatePanel 控件中包含的部分页面。UpdatePanel 控件可以嵌套。如果刷新父面板，则也会刷新所有嵌套的面板。

UpdatePanel 控件有 2 个元素，即<ContentTemplate>和<Triggers>元素，它们可以定义网页中引发异步回送的触发器。

#### 1. <ContentTemplate>元素

需要在异步回送中改变的内容都应包含在 UpdatePanel 控件<ContentTemplate>部分中。默认情况下，包含在 UpdatePanel 控件<ContentTemplate>部分中的任何回发控件（如按钮、单选按钮、列表框等）都将导致异步回送并刷新部分页内容。

#### 2. <Triggers>元素



在【例 11-1】中，由于标签和按钮控件都包含在 UpdatePanel 控件<ContentTemplate>部分中，所以当发生异步回送时，不仅回送了标签控件的内容，而且还回送了按钮的所有代码，这增加了网络传输的异步请求和响应的数据量。理想的情况是应该仅仅回送标签控件的内容，也就是说应在 UpdatePanel 控件<ContentTemplate>部分中只包含标签控件，不包含按钮控件，但是如何将 UpdatePanel 控件外的按钮控件定义为引发异步回送的触发器呢？解决的方法是通过配置<Triggers>元素来定义触发器。

<Triggers>元素有 2 个控件：AsyncPostBackTrigger 和 PostBackTrigger。PostBackTrigger 会进行完整页面的回送，而 AsyncPostBackTrigger 仅执行异步回送。

**【例 11-2】** 优化【例 11-1】，体现<Triggers>元素的应用。

(1) 在网站“Chapter11”中添加一个网页，命名为“example11-2.aspx”。

(2) 按照设计 example11-1.aspx 的方法来设计 example11-2.aspx，这里不再详述，请参考【例 11-1】。

(3) 将 Button 控件移到 UpdatePanel 控件外，打开 UpdatePanel 控件的属性窗口，单击 Triggers 属性后的图标，系统弹出“UpdatePanelTriggers 集合编辑器”对话框，单击【添加】按钮中的图标，选择“AsyncPostBackTrigger”，ControlID 选择“Button1”，EventName 选择“Click”，如图 11.5 所示。

(4) 按【Ctrl+F5】组合键运行网页，单击【获取】按钮，AJAX 获取的时间在改变，如图 11.6 所示。

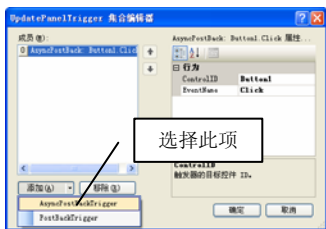


图 11.5 UpdatePanelTriggers 集合编辑器的编辑



图 11.6 网页运行的结果

## 程序分析：

与【例 11-1】不同的是，当发生异步回送时，仅回送了标签控件的内容，不回送按钮的代码，这样就减少了网络传输的异步请求和响应的数据量。

## 3. 与UpdatePanel控件不兼容的控件

下面的 ASP.NET 控件与部分页更新不兼容，因此不应在 UpdatePanel 控件内使用。

(1) 处于多种情况下的 TreeView 控件。一种情况是启用了不是异步回发的一部分的回调。另一种情况是将样式直接设置为控件属性，而不是通过使用对 CSS 样式的引用来隐式设置控件的样式。还有一种情况是 EnableClientScript 属性为 false（默认值为 true）。另外，还有一种情况是在异步回发之间更改了 EnableClientScript 属性的值。

(2) Menu 控件（将样式直接设置为控件属性，而不是通过使用对 CSS 样式的引用来隐式设置控件的样式时）。

(3) FileUpload 和 HtmlInputFile 控件（当它们作为异步回发的一部分用于上载文件时）。

(4) GridView 和 DetailsView 控件（当它们的 EnableSortingAndPagingCallbacks 属性设置为 true 时，默认值为 false）。

(5) Login、PasswordRecovery、ChangePassword 和 CreateUserWizard 控件（其内容尚未转换为可编辑的模板）。

(6) Substitution 控件。

若要在 UpdatePanel 控件内使用 FileUpload 或 HtmlInputFile 控件，请将提交文件的回发控件设置为面板的 PostBackTrigger 控件。仅可以在回发方案中使用 FileUpload 和 HtmlInputFile 控件。

所有其他控件都可以在 UpdatePanel 控件内发挥作用。不过，在某些情况下，控件在 UpdatePanel 控件内可能不会按预期方式工作。这些情况包括：

(1) 通过调用 ClientScriptManager 控件的注册方法注册脚本；

(2) 在该控件呈现过程中直接呈现脚本或标记，例如，通过调用 Write 方法。

## 11.3.4 Timer控件

在 ASP.NET 页面上进行异步回送时，一个常见的任务是希望这些异步回送以特定的时间间隔进行，此时可以使用 Timer 控件。如果将 Timer 控件用于 UpdatePanel 控件，则可以按定义的时间间隔启用部分页更新。

Timer 控件有 2 个主要的属性。

(1) Interval 属性：用于指定回送发生的频率，以毫秒为单位，其默认值为 60 000 毫秒



(即 60 秒)。


(2) Enabled 属性：用于打开或关闭 Timer，默认值为 True。

若回送是由 Timer 控件启动的，则 Timer 控件将在服务器上引发 Tick 事件。当页发送到服务器时，可以创建 Tick 事件的事件处理程序来执行一些操作。

**【例 11-3】** 定时回送刷新部分时间显示。

(1) 在网站“Chapter11”中添加一个网页，命名为“example11-3.aspx”。

(2) 切换到设计视图，从工具箱的 AJAX Extensions 组中拖放一个 ScriptManager 控件到页面上。再放一个 Label 控件，ID 为“Label1”，并在前面输入“服务器时间：”提示字样，如图 11.7 所示。

(3) 从工具箱的 AJAX Extensions 组中拖放一个 UpdatePanel 控件到页面上，在 UpdatePanel 控件中放入一个 ID 为“Lable2”的 Lable 控件和一个 Timer 控件。在 Lable2 的前面输入“利用 AJAX 获取服务器时间：”提示字样，Timer 的 Interval 属性设置为“1000”。单击 图标，双击 Tick 后的空白处，添加 Click 事件代码：

```
protected void Timer1_Tick(object sender, EventArgs e)
{
    Label2.Text = DateTime.Now.ToString();
}
```

(4) 在 Page\_Load 方法体内添加代码，如下所示：

```
Label1.Text = DateTime.Now.ToString();
```

(5) 按 **【Ctrl+F5】** 组合键运行网页，结果如图 11.8 所示。



图 11.7 页面设计



图 11.8 网页运行结果

**程序分析：**

运行页面后，会看到页面上的利用 AJAX 获取服务器时间每隔 1 秒刷新一次，回送时没有整页回送，而是异步回送。

当 Timer 控件在 UpdatePanel 控件外部时，必须将 Timer 控件显式定义为要更新的 UpdatePanel 控件的触发器。通过设置 UpdatePanel 控件的 Triggers 属性集合中的 AsyncPostBackTrigger 控件及其 ControllID 和 EventName 属性的值，来定义异步回送的触发器为 Timer 控件。

**注意：**

(1) 使用 Timer 控件时，必须在网页中包括 ScriptManager 类的实例。

(2) 如果不同的 UpdatePanel 控件需要以不同的时间间隔更新，则可以在网页上包含多个 Timer 控件。或者，可以将 Timer 控件的单个实例用做网页中多个 UpdatePanel 控件的触发器。

(3) 将 Timer 控件的 Interval 属性设置为一个较小值会产生发送到 Web 服务器的大量通信。



### 11.3.5 UpdateProgress控件

当一些异步回送需要执行一段时间时，由于响应比较大或者获得结果和发送回客户机所需的计算时间较长，所以客户等待的时间就较长。UpdateProgress 控件为客户机提供了一个可视化的指示器，显示部分页更新的进度情况。

**【例 11-4】** 使用 UpdateProgress 显示部分页的更新进度。

(1) 在网站“Chapter11”中添加一个网页，命名为“example11-4.aspx”。

(2) 切换到设计视图，从工具箱的 AJAX Extensions 组中拖放 1 个 ScriptManager 控件、1 个 UpdateProgress 控件、1 个 Button 控件和 1 个 UpdatePanel 控件到页面上。再放 1 个 Label 控件到 UpdatePanel 控件中，在 UpdateProgress 中输入“正在更新...”字样，Button 的 Text 属性设置为“更新”，如图 11.9 所示。

(3) 打开 UpdatePanel 控件的属性窗口，单击 Triggers 属性后的图标，系统弹出“UpdatePanelTriggers 集合编辑器”对话框，单击【添加】按钮中的图标，选择“AsyncPostBackTrigger”，ControlID 选择“Button1”，EventName 选择“Click”。

(4) 切换到设计视图，双击 Button 控件，添加 Button1 的 Click 事件代码，如下所示：

```
System.Threading.Thread.Sleep(10000);           //线程休眠 10 秒
Label1.Text = "更新完成!";
```

(5) 按【Ctrl+F5】组合键运行网页，单击【更新】按钮，更新时的页面如图 11.10 所示，更新后的页面如图 11.11 所示。

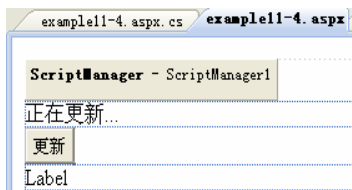


图 11.9 页面设计



图 11.10 更新时的页面



图 11.11 更新后的页面

有些时候我们不希望立即显示进度信息，而是希望等待一段时间后再显示。这可以通过设置 UpdateProgress 控件的 DisplayAfter 属性来实现，DisplayAfter 属性值的单位是毫秒。

**注意：**还有很多支持AJAX特定功能的服务器控件，由于微软把它们当做开放源代码项目，所以没有把它们集成到VS 2008 中。可以到 <http://www.asp.net/AJAX> 网站链接下载 ASP.NET AJAX控件工具集。

## 习 题

1. 什么是 AJAX？为什么使用 AJAX？
2. 如何建立 ASP.NET AJAX 应用程序？

# 第二部分 实 验

## 实验 1 创建与发布ASP.NET应用程序

### 1. 实验目的


- (1) 熟悉 ASP.NET 运行环境。
- (2) 熟悉 ASP.NET 应用程序的创建。
- (3) 了解常用控件。
- (4) 掌握 Web 页的发布。

### 2. 实验内容

设计简单的注册网站并发布此网站。

### 3. 实验步骤

设计步骤如下：

- (1) 打开 VS 2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，如图 T1.1 所示，创建一个使用文件系统的 ASP.NET 网站，单击【确定】按钮。
- (2) VS 2008 开发环境弹出 Default.aspx 页面的源视图，单击页面底部的  设计图标，切换到设计视图。
- (3) 单击菜单栏“表”→“插入表”，弹出“插入表格”对话框，在对话框的“行数”中输入“6”，单击【确定】按钮，在页面上添加了一个 6 行 2 列的表格。
- (4) 打开“工具箱”，从“标准”栏中拖放 5 个 Label、3 个 TextBox、1 个 RadioButtonList、1 个 DropDownList 和 2 个 Button 控件到表格中。控件的布局如图 T1.2 所示。其中控件的属性设置如表 T1.1 所示。

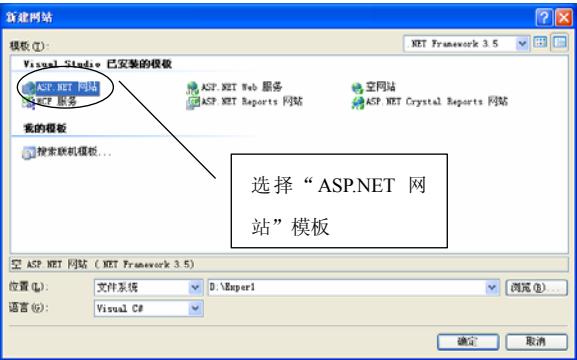


图 T1.1 “新建网站”对话框

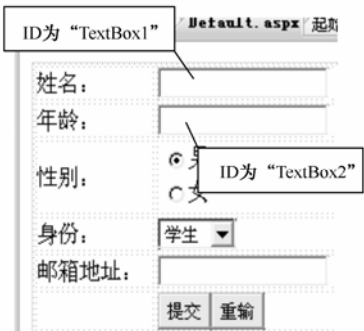


图 T1.2 控件布局

表 T1.1 控件的属性设置

控件 ID	Text 属性值	控件 ID	Text 属性值
Label1	姓名:	Label5	邮箱地址:
Label2	年龄:	Button1	提交
Label3	性别:	Button2	重输
Label4	身份:		

其中 RadioButtonList 和 DropDownList 控件的设置如下所述。



- ① 打开 RadioButtonList 控件的属性窗口→单击 Items 后面的图标，弹出“ListItem 集合编辑器”对话框，在此对话框中单击【添加】按钮，属性设置如图 T1.3 所示，其中成员“男”的 Selected 属性设置为“True”，“女”的 Selected 属性设置为“False”。
- ② 打开 DropDownList 控件的属性窗口→单击 Items 后面的图标，弹出“ListItem 集合编辑器”对话框，在此对话框中单击【添加】按钮，属性设置如图 T1.4 所示，其中成员“学生”的 Selected 属性设置为“True”，其他成员的 Selected 属性设置为“False”。



图 T1.3 RadioButtonList 控件集合编辑器



图 T1.4 DropDownList 控件集合编辑器

(5) 双击【提交】按钮，添加 Button1 的 Click 事件代码（阴影部分），如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    string s = null; //定义字符串变量 s 并清空
    s =Label1.Text + TextBox1.Text; // s← “姓名:” +姓名后面文本框的内容
    s += Label2.Text + TextBox2.Text; // s←s+ “年龄:” +年龄后面文本框的内容
    s += Label3.Text + RadioButtonList1.SelectedValue; // s←s+ “性别:” +性别选择的内容
    s += Label4.Text + DropDownList1.SelectedValue; // s←s+ “身份:” +身份选择的内容
    s += Label5.Text + TextBox3.Text; // s←s+ “邮箱地址:” +邮箱地址后面文本框的内容
}
```

(6) 双击【重输】按钮，添加 Button2 的 Click 事件代码（阴影部分），如下所示：

```
protected void Button2_Click(object sender, EventArgs e)
{
    TextBox1.Text = null; // 给第一个文本框清空
    TextBox2.Text = null; // 给第二个文本框清空
}
```

```

        TextBox3.Text = null; // 给第三个文本框清空
        Response.Write("<script>alert('"+s+"')</script>"); // 显示 s 中的内容
    }
}
```

(7) 按【Ctrl+F5】组合键运行网页，输入信息，单击【提交】按钮，结果如图 T1.5 所示。

**注意：**代码的含义在以后的章节中系统介绍，这里直接输入即可。

(8) 发布网站。操作步骤如下所示：

① 这里选择 VS 2008 中提供的预编译部署功能进行简单的网站部署测试。在 VS 2008 中选择“生成”→“发布网站”，弹出如图 T1.6 所示的对话框，选择网站发布的目标位置，单击【确定】按钮。

② 目标文件夹即是最终的网站应用程序，可以将其直接部署（即复制）到 Web 服务器中。

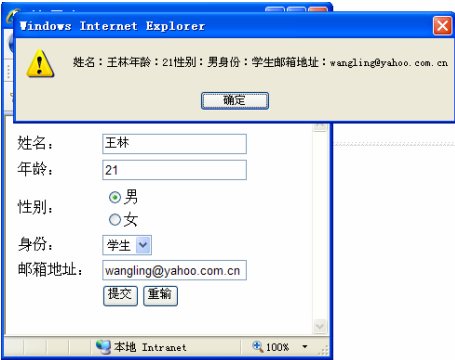


图 T1.5 注册网页的运行结果

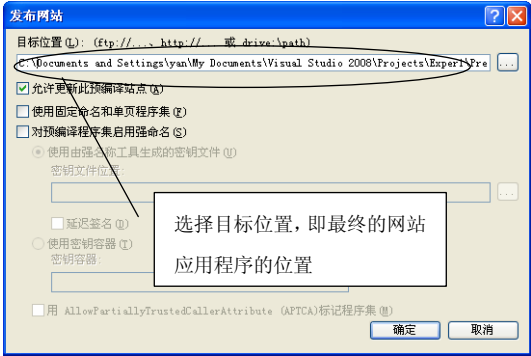


图 T1.6 “发布网站”对话框

4. 思考与练习

- (1) 如果操作系统未带有 IIS，则下载 IIS 并安装。
- (2) 练习把此网站发布到本地 IIS 中，并在浏览器中打开。

**提示：**思考与练习中有本教材未涉及的部分，请同学们查看相关资料来完成，以此提高 ASP.NET 的知识范围和培养查阅资料的习惯。

## 实验 2 C#语言基础应用

### 1. 实验目的

- (1) 掌握 C#语言基础部分的应用。
- (2) 熟悉常用控件的应用。
- (3) 掌握 ASP.NET 应用程序的创建。


### 2. 实验内容

通过设计一个简单的计算器来掌握 C#语言基础部分的应用。

### 3. 实验步骤

设计步骤如下所示：

(1) 打开 VS 2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，创建一个使用文件系统的 ASP.NET 网站，文件存放的位置是“D:\Exper2”（Exper2 文件夹是已经新建的），单击【确定】按钮。

(2) VS 2008 开发环境弹出 Default.aspx 页面的源视图，单击页面底部的  设计图标，切换到设计视图。从工具箱的“标准”栏中拖放 1 个 TextBox 控件和 1 个 Button 控件到页面上，以此页面上的 Button 控件为基础，复制 15 个 Button 控件，控件的布局如图 T2.1 所示。按图 T2.1 修改各个 Button 控件的 Text 属性，其中 ID 为“Button13”的 Text 属性设置为“=”。TextBox 的 ReadOnly 属性设置为“True”。

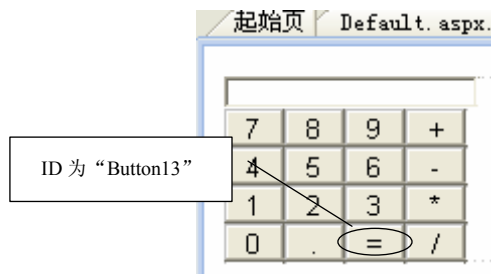


图 T2.1 页面设计部分截图

(3) 双击 ID 为“Button13”的控件，在 Button13\_Click 方法体内添加如下代码（阴影部分）：

```
protected void button13_Click(object sender, EventArgs e) //单击“=”计算
{
    if (!ifFirst)
    {
        return;
    }
    //返回，不执行以下程序
}
```

```

for (int i = 0; i < luruNumber; i++) //查找出数组中有多少运算符
{
    if (inputNumber[i] == "+" || inputNumber[i] == "-" || inputNumber[i] == "*" ||
        inputNumber[i] == "/")
    { operNumber++; }
}
double[] caoZuNumber = new double[operNumber + 1]; //定义存放运算数据的数组
string[] caoZuMark = new string[operNumber]; //定义存放运算符的数组
string strnumber = null; //定义字符串类型的运算数据变量
int j = 0; //第几个操作符
for (int i = 0; i < luruNumber; i++) //把输入的字符转变成数据并放入数组中，操作符号
    也放入数组中
{
    switch (inputNumber[i])
    {
        case "0":
        case "1":
        case "2":
        case "3":
        case "4":
        case "5":
        case "6":
        case "7":
        case "8":
        case "9":
        case ".":
            strnumber += inputNumber[i]; //把字符连接成字符串
            break;
        case "+":
            caoZuNumber[j] = Convert.ToDouble(strnumber); //字符串类型转换为数据类型
            strnumber = null;
            caoZuMark[j] = "+";
            j++;
            break;
        case "-":
            caoZuNumber[j] = Convert.ToDouble(strnumber);
            strnumber = null;
            caoZuMark[j] = "-";
            j++;
            break;
        case "*":
            caoZuNumber[j] = Convert.ToDouble(strnumber);
            strnumber = null;
            caoZuMark[j] = "*";
            j++;
            break;
        case "/":

```

```

        caoZuNumber[j] = Convert.ToDouble(strnumber);
        strnumber = null;
        caoZuMark[j] = "/";
        j++;
        break;
    }
}

```

(4) 在 Default.aspx.cs 中添加其他代码，如下所示（阴影部分）：


```
protected void Page_Load(object sender, EventArgs e)
```

```

{
    this.Title = "简单计算器"; //定义标题
    if (!Page.IsPostBack) //第一次加载页面时执行下面代码
    {
        TextBox1.Text = null; //textBox1 控件的 Text 值被设置为空
        TextBox1.Style.Add("text-align", "right"); //TextBox 控件从右边显示数据
    }
}

static int luruNumber = 0; //输入的字符的个数
static string[] inputNumber = new string[100]; //定义一个数组用于存放输入的字符
static int operNumber = 0; //计算符号的个数
static double result = 0; //算出的结果
static bool ifFirst = true; //是否第一次计算，此计算器不允许把结果用于下一次的计算
protected void number_Click(object sender, EventArgs e) //数据按钮
{
    Button btn_select = (Button)sender; //拆箱操作
    if (ifFirst) //判断是否是第一次运算，如果是第一次运算则执行以下程序
    {
        TextBox1.Text += btn_select.Text; //显示单击按钮所输入的字符
        if (TextBox1.Text == ".")
        {
            TextBox1.Text = "0.";
        }
        inputNumber[luruNumber] = btn_select.Text; //把单击的按钮控件的 Text 值存放在数组中
        luruNumber++;
    }
    else //如果不是第一次运算则执行以下程序
    {
        TextBox1.Text = btn_select.Text;
        if (TextBox1.Text == ".")
        {
            TextBox1.Text = "0.";
        }
        inputNumber[luruNumber] = btn_select.Text;
        luruNumber++;
        ifFirst = true;
    }
}
}

```

(5) 单击页面底部的  设计图标，切换到设计视图。打开 Text 为“7”的 Button 控件属



性窗口→右击⚡图标→选择 Click 事件响应方法 “number\_Click”。除了 ID 为 “Button13” 的 Button 控件，其他的 Button 控件的 Click 事件响应方法都为 “number\_Click”。

（6）按【Ctrl+F5】组合键运行网页，如图 T2.2 所示。读者试试此计算器。

**注意：**此计算器只执行从前向后的运算， $4+3*5$  其实为  $(4+3)*5$ 。

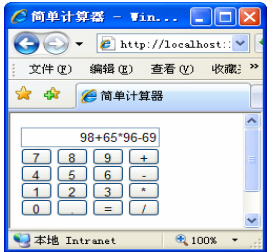


图 T2.2 简单计算器

5. 思考与练习

- （1）添加一个按钮，实现撤销最后一个输入数字。
- （2）添加一个按钮，实现清空输入或者计算结果。

# 实验 3 面向对象编程

## 1. 实验目的

- (1) 掌握 C#语言面向对象程序设计。
- (2) 掌握 ASP.NET 应用程序的创建。




## 2. 实验内容

通过设计计算图形面积页面来掌握 C#语言面向对象程序设计。


## 3. 实验步骤

设计步骤如下：

(1) 打开 VS 2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，创建一个使用文件系统的 ASP.NET 网站，文件存放的位置是“D:\Exper3”（Exper3 文件夹是已经新建的），单击【确定】按钮。

(2) VS 2008 开发环境弹出 Default.aspx 页面的源视图，单击页面底部的  设计图标，切换到设计视图。从工具箱的标准栏中拖放 1 个 RadioButtonList（图标为  RadioButtonList）、1 个 TextBox、1 个 Button 和 3 个 Panel 控件（图标为  Panel）到页面上。Button 控件的 Text 属性设置为“计算”，Panel2 和 Panel3 的 Visible 设置为“False”。

**注意：**RadioButtonList 为单选按钮列表，用户在这一组列表项中只能选择一项；Panel 为容器控件，作为其他控件的父控件。这里只需了解，在下面章节中详细介绍。

(3) 打开 RadioButtonList 控件的属性窗口，单击 Items 后面的  图标，弹出“ListItem 集合编辑器”对话框，在此对话框中单击【添加】按钮，属性设置如图 T3.1 所示，其中成员“矩形”的 Selected 属性设置为“True”，其他成员的 Selected 属性设置为“False”。

(4) 在 3 个 Panel 控件中分别添加提示字样和 TextBox 控件用于输入图形的数据。设计后的页面如图 T3.2 所示。其中 Panel1 包含“TextBox2”和“TextBox3”，Panel2 包含“TextBox4”，Panel3 包含“TextBox5”、“TextBox6”和“TextBox7”。TextBox 的 Text 属性都设置为“0”。

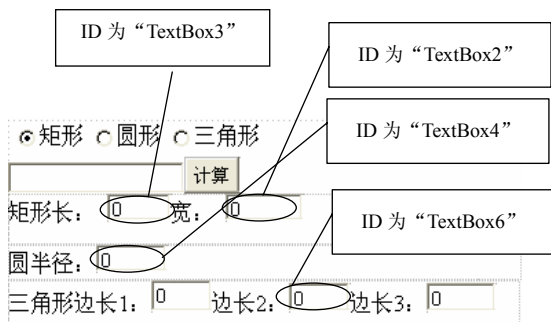
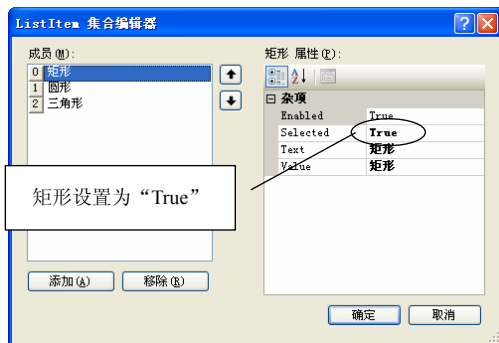



图 T3.1 RadioButtonList 控件集合编辑器

图 T3.2 设计后的页面部分截图

(5) 单击 RadioButtonList 控件右上角的图标，选择“启用 AutoPostBack”复选框，如图 T3.3 所示。表示当单击单选框时向服务器提交了一次事件。

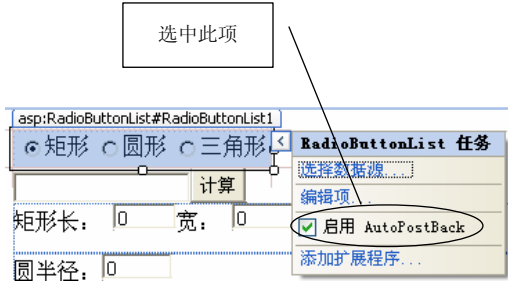


图 T3.3 启用 AutoPostBack

(6) 双击 RadioButtonList 控件，添加 Click 事件代码，如下所示（阴影部分）：

```
protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e)
{
    string switchshape=RadioButtonList1.Selected.Value;           //所选择的图形
    if (switchshape == "矩形")                                     //如果是矩形则 Panel1 可见
    {
        Panel1.Visible = true;
        Panel2.Visible = false;
        Panel3.Visible = false;
    }
    else if(switchshape=="圆形")                                   //如果是矩形则 Panel2 可见
    {
        Panel1.Visible = false;
        Panel2.Visible = true;
        Panel3.Visible = false;
    }
    else if(switchshape=="三角形")                                 //如果是矩形则 Panel3 可见
    {
        Panel1.Visible = false;
        Panel2.Visible = false;
        Panel3.Visible = true;
    }
    TextBox1.Text = null;                                           //TextBox1 的 Text 清空
}
```

(7) 切换到设计视图，双击 Button 控件，添加 Button 控件的 Click 事件代码，如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    switch (RadioButtonList1.Selected.Value)
    {
        case "圆形":                                               //如果是圆形，则计算面积
            Circle C = new Circle(Convert.ToDouble(TextBox4.Text));
```

```

        TextBox1.Text = C.GetArea().ToString();
        break;
    case "矩形":
        //如果是矩形，则计算面积
        Rectangular R = new Rectangular(Convert.ToDouble(TextBox2.Text),
            Convert.ToDouble(TextBox3.Text));
        TextBox1.Text = R.GetArea().ToString();
        break;
    case "三角形":
        //如果是三角形，则计算面积
        Triangle T = new Triangle(Convert.ToDouble(TextBox5.Text),
            Convert.ToDouble(TextBox6.Text), Convert.ToDouble(TextBox7.Text));
        TextBox1.Text = T.GetArea().ToString();
        break;
    }
}

```

(8) 在\_Default 局部类的下面添加图形类，代码如下所示：

```

public class Shape
{
    //定义图形类 Shape
    protected const double PI = 3.14;
    //定义圆周率
    public Shape()
    //无参构造函数
    { }
    public virtual double GetArea()
    //定义虚方法，计算面积
    { return 0.0; }
}
public class Circle : Shape
//定义圆类 Circle，从 Shape 类中派生
{
    private double Radius;
    //半径
    public Circle(double R)
    //圆的构造函数
    { this.Radius = R; }
    public override double GetArea()
    //对虚方法重载，计算圆的面积
    { return PI * Radius * Radius ; }
}
public class Rectangular : Shape
//定义矩形类Rectangular,从Shape类中派生
{
    protected double Length, Width;
    //矩形的长和宽
    public Rectangular(double L, double W)
    //矩形的构造函数
    {
        this.Length = L;
        this.Width = W;
    }
    public override double GetArea()
    //对虚方法重载，计算矩形的面积
    { return Length * Width; }
}
public class Triangle : Shape
//定义三角形类 Triangle,从 Shape 类中派生
{

```

```

protected double LengthOne, LengthTwo, LengthThree; //三角形的三条边
public Triangle()
{ LengthOne = LengthTwo = LengthThree = 0; }
public Triangle(double lo, double lt, double lth) //三角形的构造函数
{
    this.LengthOne = lo;
    this.LengthTwo = lt;
    this.LengthThree = lth;
}
public override double GetArea() //对虚方法重载, 计算三角形的面积
{
    double P = (LengthOne + LengthTwo + LengthThree) / 2;
    // System.Math.Sqrt 为计算指定数据的平方根
    return System.Math.Sqrt(P*(P-LengthOne)*(P-LengthTwo)*(P-LengthThree));
}
}

```

(9) 按【Ctrl+F5】组合键运行网页，计算机圆的面积，结果如图 T3.4 所示。



图 T3.4 计算半径为 50 的圆的面积

#### 4. 思考与练习

- (1) 为图形添加颜色属性。
- (2) 添加计算周长功能。

## 实验 4 内置对象的应用

### 1. 实验目的

- (1) 掌握内置对象的创建与应用。
- (2) 掌握 ASP.NET 应用程序的起始页的创建。


### 2. 实验内容

通过设计简单的聊天系统来掌握 Session 和 Application 内置对象的应用。

### 3. 实验步骤

设计步骤如下：

(1) 打开 VS 2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，创建一个使用文件系统的 ASP.NET 网站，文件存放的位置是“D:\Exper4”（Exper4 文件夹是已经创建过的），单击【确定】按钮。

(2) 打开“解决方案资源管理器”→右击站点→选择“添加新项”，在弹出的“添加新项”对话框中选择“全局应用程序类”，图标为全局应用程序类。在 Global.asax 文件中添加代码，如下所示（黑体部分）：

```
void Application_Start(object sender, EventArgs e)
{
    // 在应用程序启动时运行的代码
    Application["chatcontent"] = "<h2>欢迎来到聊天室!</h2>"; //定义存放聊天记录变量
}
void Application_End(object sender, EventArgs e)
{
    //在应用程序关闭时运行的代码
    Application.RemoveAll(); //移除所有对象
}
```

(3) 打开“解决方案资源管理器”→右击站点→选择“添加新项”选项，在弹出的“添加新项”对话框中选择“Web 窗体”模板→命名为“login.aspx”→单击【确定】按钮。

按照此方法分别添加“main.aspx”和“send.aspx”Web 窗体。其中“login.aspx”是登录页面，“main.aspx”是显示聊天内容页面，“send.aspx”是发送聊天内容页面。

(4) 创建一个 HTML 页面把所有的页面组合成一个页面。打开“解决方案资源管理器”→右击站点→选择“添加新项”选项，在弹出的“添加新项”对话框中选择“HTML 页”模板→默认名为“HTMLPage.htm”→单击【确定】按钮。修改此页面的 HTML 代码，修改后的代码如下所示：

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
    <title>聊天系统</title>
</head>
<frameset rows="80%,20%" >                //窗口被分成上下两部分,各占页面的 80%和 20%
<frame src="main.aspx"></frame>            //上面窗口引用 main.aspx
<frame src="send.aspx"></frame>            //下面窗口引用 send.aspx
</frameset>
<body>
</body>
</html>

```

(5) 设计“login.aspx”页面。打开“login.aspx”的设计视图，从工具箱的标准栏中拖放 2 个 TextBox 控件和 1 个 Button 控件到此页面上，TextBox2 的 TextMode 属性设置为“Password”，Button1 的 Text 属性设置为“登录”，添加一些提示字样，设计后的页面如图 T4.1 所示。双击 Button 控件，添加事件代码，如下所示：

```

protected void Button1_Click(object sender, EventArgs e)
{
    if(TextBox1.Text.Trim()==string.Empty)    //如果未输入用户名则返回
    {
        Response.Write("<script>alert('请输入用户名!')</script>");
        return;
    }
    if(TextBox2.Text!="123456")                //如果密码不为规定的 123456 则返回
    {
        Response.Write("<script>alert('密码不正确,请正确输入本聊天室的密码!')</script>");
        return;
    }
    Session["username"] = TextBox1.Text;      //保存 Session 变量 username
    Response.Redirect("send.aspx");           //跳转到 send.aspx 页面
}

```

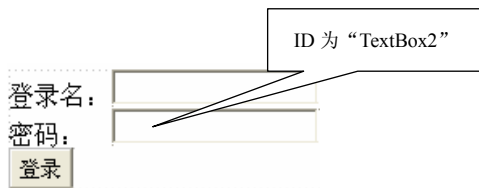


图 T4.1 login.aspx 页面设计

(6) 设计“send.aspx”页面。打开“send.aspx”的设计视图，从工具箱的标准栏中拖放 1 个 TextBox 控件、1 个 Label 控件和 2 个 Button 控件到此页面上，Button1 的 Text 属性设置为“发送”，Button2 的 Text 属性设置为“重新登录”，设计后的页面如图 T4.2 所示。双击【发送】按钮，添加事件代码，如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    string message;
    message = "<font color='blue'>" + Session["username"].ToString() + "</font>说: ";//登录者字体为蓝色
    message += TextBox2.Text;
    message += "(<i>" + DateTime.Now.ToString()+"</i>");//发送的时间设置为斜体
    message += "<br>";
    Application.Lock();                                //锁定，不允许其他用户修改变量
    Application["chatcontent"] = (string)Application["chatcontent"] + message;//聊天内容放在对象中
    Application.UnLock();                              //取消锁定，允许其他用户修改变量
    TextBox2.Text = null;
}
}
```

切换到设计视图，双击【重新登录】按钮，添加 Click 事件代码，如下所示：

```
protected void Button2_Click(object sender, EventArgs e)
{
    Response.Redirect("login.aspx");                    //当重新登录时重定向到登录页面
}
}
```

当此页面运行时判断是否登录，在 Page\_Load 方法体内添加代码，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["username"] != null)                    //如果已经登录，Label 显示登录者说：
    { Label1.Text = Session["username"].ToString() + "说: "; }
    else
    { Response.Redirect("login.aspx"); }
}
}
```



图 T4.2 send.aspx 页面设计

(7) 打开 “main.aspx.cs” 代码页，在 Page\_Load 方法体内添加代码，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.AddHeader("Refresh","3");                  //每 3 秒刷新页面
    Response.Write((string)Application["chatcontent"]); //在页面上输出聊天内容
}
}
```

(8) 设置起始页。打开 “解决方案资源管理器” → 右击 “HTMLPage.htm” 文件 → 选择 “设为起始页” 选项，表示首先运行的页面，如图 T4.3 所示。

(9) 按【Ctrl+F5】组合键运行此网站，结果如图 T4.4 所示。



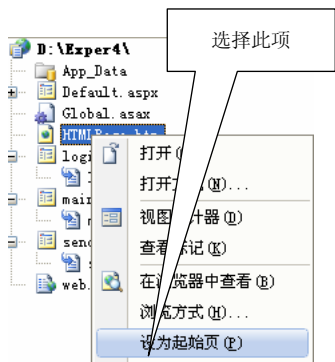


图 T4.3 设置起始页

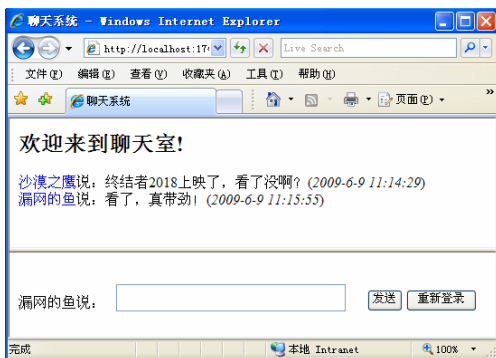


图 T4.4 聊天系统

**注意：**此聊天系统是在知道密码的情况下进入聊天室聊天的。

#### 4. 思考与练习

添加显示所有进入聊天室的用户的功能。

# 实验 5 ASP.NET服务器控件应用

## 1. 实验目的

- (1) 掌握 ASP.NET 服务器控件的主要属性、方法、事件及其应用。
- (2) 掌握 ASP.NET 验证控件的应用。

## 2. 实验内容

通过设计简单的注册页面来掌握服务器控件的使用。

## 3. 实验步骤

设计步骤如下：


- (1) 打开 VS 2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，创建一个使用文件系统的 ASP.NET 网站，文件存放的位置是“D:\Exper5”（Exper5 文件夹是已经创建过的），单击【确定】按钮。
- (2) 切换到 Default.aspx 页的设计视图，从工具箱的标准栏中拖放一个 Wizard 控件到页面上，打开 Wizard 控件的属性窗口，单击 WizardSteps 后的图标，编辑“WizardStep 集合编辑器”，如图 T5.1 所示。
- (3) 将 Wizard 控件拉到一定的大小，右击“Wizard”→选择“编辑模板”→“HeaderTemplate”。在头模板“HeaderTemplate”中输入“用户注册向导”，如图 T5.2 所示。右击“HeaderTemplate”→选择“结束模板编辑”。



图 T5.1 编辑 WizardStep 集合编辑器



图 T5.2 编辑 HeaderTemplate

- (4) “第一步”设计。从工具箱的标准栏中拖放 4 个 TextBox 控件、3 个 Label 控件，从验证栏中拖放 2 个 RequiredFieldValidator 控件和 1 个 CompareValidator 控件放在 Wizard 控件

上。3 个 Label 控件的 Text 属性设置为 “\*”，表示必须填写，验证控件的属性设置如表 T5.1 所示，TextBox3 和 TextBox4 的 TextMode 设置为 “Password”。设计后的页面如图 T5.3 所示。

表 T5.1 第一步向导控件设置

控件 ID	属性: 属性值
RequiredFieldValidator1	ErrorMessage: 请输入用户名; ControlToValidate: TextBox1
RequiredFieldValidator2	ErrorMessage: 请输入密码; ControlToValidate: TextBox2
CompareValidator1	ErrorMessage: 密码不一致; ControlToCompare: TextBox2; ControlToValidate: TextBox3

**注意：**页面上的提示字符请读者按照页面添加。

(5) “第二步”设计。单击【第二步】，从工具箱的标准栏中拖放 1 个 TextBox 控件、1 个 RadioButtonList 控件、1 个 ListBox 控件，从验证栏中拖放 1 个 RegularExpressionValidator 控件放在 Wizard 控件上。RadioButtonList 和 ListBox 控件都要编辑 Items 属性，如表 T5.2 所示。设计后的页面如图 T5.4 所示。

表 T5.2 第二步向导控件设置

控件 ID	属 性 设 置
RadioButtonList1	ListItem 集合编辑器添加成员 “男”、“女”，男成员 Selected 设置为 “True”
ListBox1	ListItem 集合编辑器添加成员 “学生”、“工人”、“农民”、“教师”
RegularExpressionValidator1	ErrorMessage: 身份证号码格式不对; ControlToCompare: TextBox5; ValidationExpression: 中华人民共和国身份证号码 (ID 号)

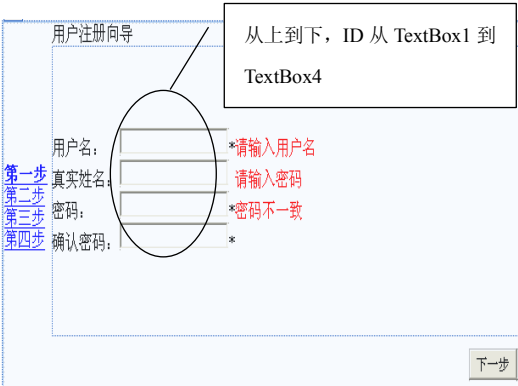


图 T5.3 “第一步”设计

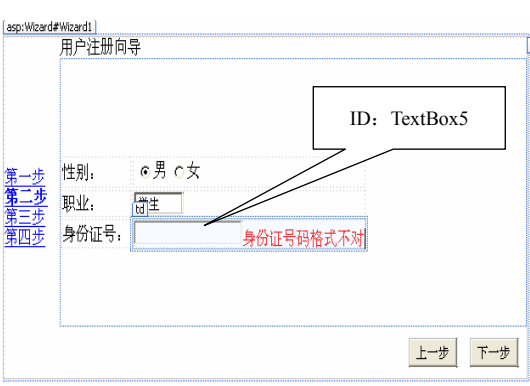


图 T5.4 “第二步”设计

(6) “第三步”设计。单击【第三步】，从工具箱的标准栏中拖放 2 个 TextBox 控件、1 个 Label 控件、3 个 DropDownList 控件放在 Wizard 控件上，再从验证栏中拖放 1 个 RegularExpressionValidator 控件和 1 个 RequiredFieldValidator 控件放在 Wizard 控件上，控件的属性设置如表 T5.3 所示。设计后的页面如图 T5.5 所示。

(7) “第四步”设计。单击【第四步】，从工具箱的标准栏中拖放 1 个 TextBox 控件放在 Wizard 控件上，ID 为 “TextBox8” 的 TextBox 控件的 TextMode 设置为 “MultiLine”，表示多行显示。设计后的页面如图 T5.6 所示。

表 T5.3 第三步向导控件设置

控件 ID	属 性 设 置
RequiredFieldValidator3	ErrorMessage: 请输入邮箱地址; ControlToCompare: TextBox6
RegularExpressionValidator2	ErrorMessage: E-mail 格式不对; ControlToCompare: TextBox6; ValidationExpression: Internet 电子邮件地址

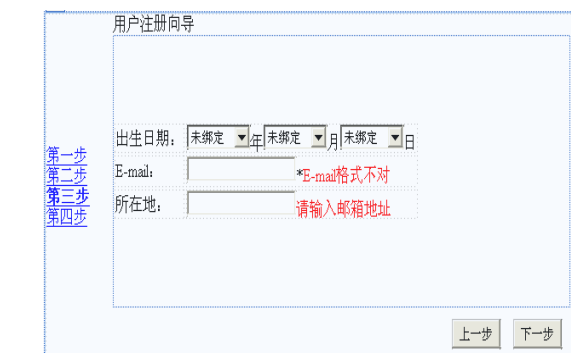


图 T5.5 第三步设计



图 T5.6 第四步设计

(8) 双击【完成】按钮，添加事件代码，如下所示（黑体部分）：

```
protected void Wizard1_FinishButtonClick(object sender, WizardNavigationEventArgs e)
{
    Response.Write("<script>alert('提交成功')</script>"); //显示“提交成功”信息框
}
```

(9) 切换到设计视图，双击【下一步】按钮，添加事件代码，如下所示：

```
protected void Wizard1_NextButtonClick(object sender, WizardNavigationEventArgs e)
{
    if(e.NextStepIndex==3) //第三步，单击下一步按钮执行下面代码
    {
        string x = ""; //存放所添信息
        x += "姓名: " + TextBox1.Text + "\n";
        x += "真实姓名: " + TextBox2.Text + "\n";
        x += "性别: " + RadioButtonList1.SelectedValue + "\n";
        x += "职业: " + ListBox1.SelectedValue + "\n";
        x += "身份证号: " + TextBox5.Text + "\n";
        x += "出生时间: " + DropDownList1.SelectedValue + "年" + DropDownList2.SelectedValue +
            "月" + DropDownList3.SelectedValue + "日" + "\n";
        x += "E-mail: " + TextBox6.Text + "\n";
        x += "所在地: " + TextBox7.Text + "\n";
        TextBox8.Text = x;
    }
}
```

(10) 当网站运行时，给 3 个 DropDownList 控件添加年月日选择时间，在 Page\_Load 方法体内添加代码，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    int i;
    string x;
    if (!IsPostBack)                                //如果第一次加载页面则执行下面代码
    {
        for (i = 1980; i <= 2000; i++)                //给 DropDownList1 添加从 1980 到 2000
        {
            x = System.Convert.ToString(i);           //将 int 类型转换为 String 类型
            DropDownList1.Items.Add(x);
        }
        for (i = 1; i <= 12; i++)                    //给 DropDownList2 添加从 1 到 12
        { DropDownList2.Items.Add(System.Convert.ToString(i)); }
        for (i = 1; i <= 31; i++)                    //给 DropDownList3 添加从 1 到 31
        { DropDownList3.Items.Add(System.Convert.ToString(i)); }
    }
}
```

(11) 按【Ctrl+F5】组合键运行网站，通过用户注册向导添加信息，单击第三步所设计页面中的【下一步】按钮后页面如图 T5.7 所示。

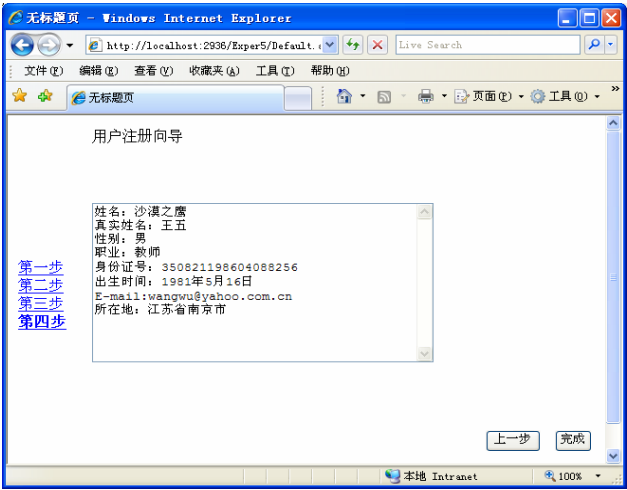


图 T5.7 第四步页面

4. 思考与练习

- (1) 在根目录下新建 Images 文件夹，新建一个用户控件，用户控件的功能是上传形象照图片文件到 Images 文件夹中。
- (2) 把“第四步”改为“第五步”，并在此之前添加“第四步”，引入此用户控件，并注册后显示形象照。

## 实验 6 母版、主题和导航设计

### 1. 实验目的

- (1) 熟练掌握母版的使用方法。
- (2) 熟练掌握主题及皮肤的使用。
- (3) 熟练掌握导航控件的使用。

### 2. 实验内容

设计一个简单的网站（ASP.NET 学习园地）来掌握母版、主题和导航控件的使用。

### 3. 实验步骤

设计步骤如下：

(1) 新建站点。打开 VS 2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，创建一个使用文件系统的 ASP.NET 网站，文件存放的位置是“D:\Exper6”（Exper6 文件夹是已经创建过的），单击【确定】按钮。

(2) 添加母版页。打开“解决方案资源管理器”→右击站点→选择“添加新项”→在“添加新项”对话框中选择“母版页”→命名为“Master1.master”→单击【确定】按钮。

(3) 添加内容页。打开“解决方案资源管理器”→右击站点→选择“添加新项”→在“添加新项”对话框中选择“Web 窗体”→命名为“sy1.aspx”→选中“选择母版页”复选框→单击【添加】按钮→在弹出的“选择母版页”对话框中选择“Master1.master”模板→单击【确定】按钮。按照此方法再分别添加“sy2.aspx”、“sy3.aspx”、“sy4.aspx”内容页。

(4) 设计母版页。打开 Master1.master 的设计视图→单击菜单“表”→“插入表”→在“插入表”对话框中设置其行列都为 2→单击【确定】按钮。合并插入表的左列，从工具箱中分别拖放 1 个 TreeView 控件和 1 个 Calender 控件放入其中。在第 2 列的上一行放入 SiteMapPath 控件，并把 ContentPlaceHolder 控件剪切、复制到第 2 列的下一行。切换到源视图，添加标题，代码如下所示（黑体部分）：

```
<h2>ASP.NET 学习园地</h2><asp:SiteMapPath ID="SiteMapPath1" runat="server"></asp:SiteMapPath>
```


代码放在 SiteMapPath 控件代码之前。打开 TreeView 控件的属性窗口，单击 Nodes 后的  图标，按照图 T6.1 所示编辑 TreeView 节点。设计后的母版页如图 T6.2 所示。



图 T6.1 TreeView 节点编辑



图 T6.2 设计后的母版页部分截图

(5) 添加站点地图。打开“解决方案资源管理器”→右击站点→选择“添加新项”→选择“站点地图”→单击【确定】按钮，添加站点地图代码，如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="sy1.aspx" title="第六章" description="第六章">
    <siteMapNode url="sy2.aspx" title="母版的学习" description="母板" />
    <siteMapNode url="sy3.aspx" title="主题的学习" description="主题" />
    <siteMapNode url="sy4.aspx" title="导航的学习" description="导航" />
  </siteMapNode>
</siteMap>
```

(6) 添加主题。打开“解决方案资源管理器”→右击站点→选择“添加 ASP.NET 文件夹”→选择“主题”，则在站点下添加了一个“App\_Themes”文件夹并在此文件夹下添加了“主题 1”。右击“App\_Themes”→选择“添加 ASP.NET 文件夹”→选择“主题”，则添加了“主题 2”。

① 设置“主题 1”。打开“解决方案资源管理器”→右击“主题 1”→选择“添加新项”→选择“外观文件”→名称为“SkinFile1.skin”→单击【确定】按钮，按照此方法再添加一个命名为“SkinFile2.skin”的外观文件。打开“Default.aspx”的设计视图→从工具箱中拖放一个 TextBox 控件到此页面上→打开 TextBox 控件的属性窗口→根据爱好设置外观属性。这里的设置如图 T6.3 所示。切换到源视图，复制 TextBox 控件代码，粘贴到 SkinFile1.skin 文件中，修改代码，修改后的代码如下所示：

```
<asp:TextBox SkinID="Skin1" runat="server" BackColor="#999966" BorderColor="Blue" Font-Bold=
"False" Font-Italic="False" Font-Overline="False" Font-Size="Smaller" Font-Strikeout="False"
Font-Underline="True" Height="195px" Width="300px" />
```

按照此方法给 SkinFile2.skin 添加代码，如下所示：

```
<asp:TextBox SkinID="Skin2" runat="server" BackColor="#999966" BorderColor="Red" Font-Bold=
"False" Font-Italic="False" Font-Overline="False" Font-Size="Smaller" Font-Strikeout="False"
Font-Underline="False" Height="195px" Width="300px" />
```

打开“解决方案资源管理器”→右击“主题 1”→选择“添加新项”→选择“样式表”→命名为“StyleSheet1.css”→单击“确定”按钮。在打开的 StyleSheet1.css 中右击空白处→选择“添加样式规则”→添加“table”元素→右击 table 元素的“{ }”中的空白处→选择“生成样式”。在“修改样式”对话框中根据爱好修改样式，样式设置后的代码如下所示：

```
table
{ background-color: #3399FF; }
```

② 设置“主题 2”。主题 2 的设置与主题 1 相同。主题 2 中添加 1 个命名为“SkinFile2.skin”的外观文件和 1 个命名为“StyleSheet2.css”的样式表。其中 SkinFile2.skin 文件代码如下所示：

```
<asp:TextBox SkinID="Skin2" runat="server" Font-Bold="True" Font-Italic="True" Font-Size="Smaller"
Font-Strikeout="False" ForeColor="#009999" Height="195px" Width="300px" />
```

StyleSheet2.css 文件代码如下所示：

```
body {}
table
{ background-color: #CCCC00; }
```

主题设计好后，打开“解决方案资源管理器”可见所添加的主题，如图 T6.4 所示。

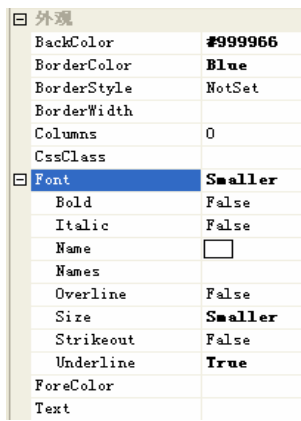


图 T6.3 TextBox 控件外观设置

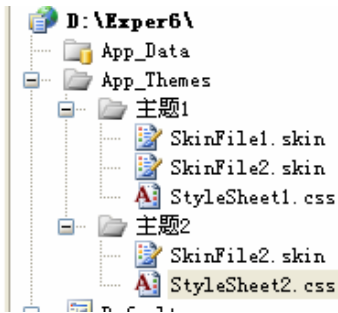


图 T6.4 所添加的主题

(7) 设置内容页。打开“sy1.aspx”内容页的设计视图，给 ContentPlaceHolder 控件添加一个 TextBox 控件。打开属性窗口→选择“DOCUMENT”→设置 Theme 属性为“主题 1”，如图 T6.5 所示。在属性窗口中选择“TextBox1”，设置 SkinID 属性为“Skin2”，TextMode 属性为“MultiLine”。打开“sy1.aspx.cs”代码页，在 Page\_Load 方法体内添加 TextBox 控件显示内容代码，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    TextBox1.Text = "本章主要介绍页面的美化";
}
```

按照此方法设置其他内容页，各个内容页的属性设置如表 T6.1 所示。“TextBox1.Text”的值读者可自己添加。



表 T6.1 各个内容页的属性设置

内容页	Theme	TextBox1 的 SkinID	TextBox1 的 TextMode
sy1.aspx	主题 1	Skin2	MultiLine
sy2.aspx	主题 1	Skin1	MultiLine
sy3.aspx	主题 2	Skin2	MultiLine
sy4.aspx			MultiLine

(8) 运行网站。按【Ctrl+F5】组合键运行网站，结果如图 T6.6 所示。



图 T6.5 选择主题

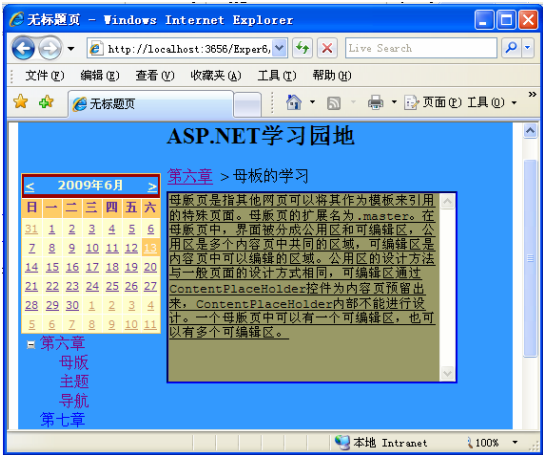


图 T6.6 网站运行的结果

#### 4. 思考与练习

- (1) 添加其他控件，通过设置主题皮肤改变控件样式。
- (2) 通过设置控件属性改变控件样式。

# 实验 7 数据库编程

## 1. 实验目的

- (1) 掌握在 SQL Server 2005 中创建数据库、表的方法。
- (2) 掌握使用 ADO.NET 编程模型访问数据库。
- (3) 掌握数据控件的使用。

## 2. 实验内容

设计一个网站注册页面，首先检测此网站是否有此注册名，如果有则返回重新注册；如果没有则把信息添加到数据库中，并显示所有注册的用户信息。

## 3. 实验步骤

设计步骤如下：

- (1) 新建站点。打开 VS 2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，创建一个使用文件系统的 ASP.NET 网站，文件存放的位置是“D:\Exper7”（Exper7 文件夹是已经创建过的），单击【确定】按钮。
- (2) 设计页面。打开 Default.aspx 页面的设计视图，从工具箱的标准栏中选择 1 个 Label 控件和 1 个 Panel 控件放到页面上，从工具箱的数据栏中选择 1 个 GridView 控件和 1 个 SqlDataSource 控件放到页面上。在 Panel 控件中插入 7 行 2 列的表，在表中拖放 5 个 TextBox 控件、1 个 DropDownList 控件和 2 个 Button 控件，并且从工具箱的验证栏中拖放 3 个 RequiredFieldValidator、1 个 CompareValidator 和 1 个 RegularExpressionValidator 验证控件，如图 T7.1 所示。
- (3) 控件属性设置。控件的属性设置如表 T7.1 所示。其中 TextBox2 和 TextBox3 的 TextMode 设置为“Password”。Button1 和 Button2 的 Text 设置为“注册”、“重输”。在表格的适当地方输入提示字样，并且 GridView1 的 Visible 设置为“False”，如图 T7.2 所示。

表 T7.1 控件的属性设置

控件 ID	属 性 设 置
Label1	Size: X-Large; Bold: True
DropDownList1	ListItem 集合编辑器添加成员“出示所在地”、“最喜欢的动物”、“母亲的生日”。其中第一个成员的 Selected: True
RegularExpressionValidator1	ErrorMessage: Email 格式不对; ControlToCompare: TextBox5; ValidationExpression: Internet 电子邮件地址
RequiredFieldValidator1	ErrorMessage: 请输入登录名; ControlToValidate: TextBox1
RequiredFieldValidator2	ErrorMessage: 请输入密码; ControlToValidate: TextBox2
RequiredFieldValidator3	ErrorMessage: 请输入 Email; ControlToValidate: TextBox5
CompareValidator1	ErrorMessage: 密码不一致; ControlToCompare: TextBox2; ControlToValidate: TextBox3

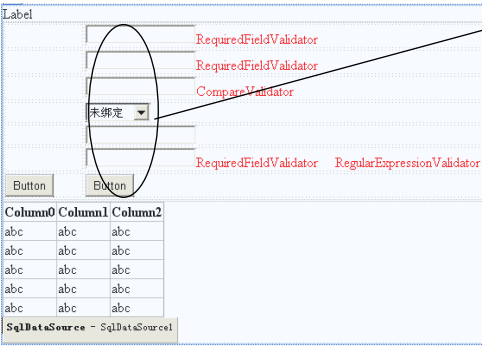


图 T7.1 页面设计

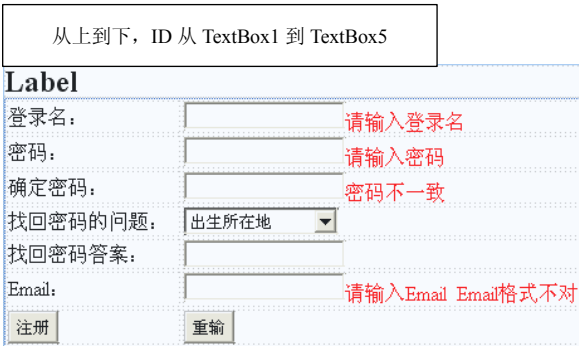


图 T7.2 控件属性设置后的页面

(4) 数据库的创建。打开“解决方案资源管理器”→右击“App\_Data”文件夹→选择“添加新项”→选择“SQL Server 数据库”→命名为“Register.mdf”→单击【添加】按钮，则在 App\_Data 文件夹下创建了 Register.mdf 数据库。

(5) 表的创建。打开“服务器资源管理器”→右击 Register 数据库下的“表”→选择“添加新表”，表的结构设置如图 T7.3 所示，关闭表的设置，命名为“LoginTable”。

列名	数据类型	允许 Null
LoginName	varchar (20)	<input type="checkbox"/>
Password	varchar (50)	<input type="checkbox"/>
Question	varchar (50)	<input checked="" type="checkbox"/>
Answer	varchar (100)	<input checked="" type="checkbox"/>
Email	varchar (50)	<input checked="" type="checkbox"/>
LoginTime	datetime	<input checked="" type="checkbox"/>


图 T7.3 表 LoginTable 的结构设置

(6) SqlDataSource1 的设置。右击“SqlDataSource1”→选择“配置数据源”→在“配置数据源”对话框中单击“新建连接”→在“添加连接”对话框中，数据源为“Microsoft SQL Server (SqlClient)”，服务器选择“69F638102711447\SQLEXPRESS”→单击【高级】按钮，高级属性的设置如表 7.2 所示。单击【测试连接】按钮测试是否可以连接。测试成功后单击【确定】按钮→单击【下一步】按钮→单击【下一步】按钮（是，将此连接另存为“RegisterConnectionString”）→在配置 Select 语句中选择“LoginName”、“Email”和“LoginTime”→单击【下一步】按钮→单击【完成】按钮。

表 7.2 高级属性的设置

属 性	属 性 值
AttachDbFilename	D:\Exper7\App_Data\Register.mdf
Integrated Security	True
User Instance	True

**注意：**69F638102711447 是机器名，读者的机器名也许不是此名称。

(7) GridView1 控件的设置。单击 GridView1 的  图标→选择“SqlDataSource1”数据源→选择“启动分页”和“启动排序”复选框→选择“编辑列”，在弹出的“字段”对话框中把

HeaderText 设置为对应的字样，这里分别为“登录名”、“Email”和“注册时间”。

(8) 注册的 Click 事件。双击【注册】按钮。添加代码，首先添加命名空间“using System.Data.SqlClient;”，代码如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    string constr = ConfigurationManager.ConnectionStrings["RegisterConnectionString"].ToString();
    SqlConnection conn = new SqlConnection(constr);
    conn.Open(); //打开数据库连接
    // 检查用户是否已存在
    SqlCommand Cmd = new SqlCommand();
    Cmd.Connection = conn;
    Cmd.CommandText = "select LoginName from LoginTable";
    SqlDataReader dr = Cmd.ExecuteReader();
    while (dr.Read())
    {
        if (dr.GetString(0) == TextBox1.Text.Trim())
        {
            Response.Write("<script>alert('登录名已经存在，请你选择另外的登录名!')</script>");
            dr.Close(); //关闭 SqlDataReader 对象
            conn.Close(); //关闭连接
            return; //有此注册名则返回
        }
    }
    dr.Close(); //关闭 SqlDataReader 对象
    //注册用户
    string SqlStr =
        "Insert into [LoginTable]([LoginName],[PassWord],[Question],[Answer],[Email],[LoginTime])
        values(@LoginName,@Password,@Question,@Answer,@Email,@LoginTime)";
    Cmd.CommandText = SqlStr;
    // 添加参数对象,并给参数赋值
    SqlParameter para1 = new SqlParameter("@LoginName", SqlDbType.VarChar, 20);
    para1.Value = TextBox1.Text.Trim();
    Cmd.Parameters.Add(para1); //Cmd 中添加参数@LoginName
    SqlParameter para2 = new SqlParameter("@Password", SqlDbType.VarChar, 50);
    para2.Value = TextBox2.Text.Trim();
    Cmd.Parameters.Add(para2); //Cmd 中添加参数@Password
    SqlParameter para3 = new SqlParameter("@Question", SqlDbType.VarChar, 100);
    para3.Value = DropDownList1.SelectedValue;
    Cmd.Parameters.Add(para3); //Cmd 中添加参数@Question
    SqlParameter para4 = new SqlParameter("@Answer", SqlDbType.VarChar, 100);
    para4.Value = TextBox4.Text.Trim();
    Cmd.Parameters.Add(para4); //Cmd 中添加参数@Answer
    SqlParameter para5 = new SqlParameter("@Email", SqlDbType.VarChar, 50);
    para5.Value = TextBox5.Text.Trim();
    Cmd.Parameters.Add(para5); //Cmd 中添加参数@Email
```

```

SqlParameter para6 = new SqlParameter("@LoginTime", SqlDbType.DateTime);
para6.Value = DateTime.Now;
Cmd.Parameters.Add(para6);           //Cmd 中添加参数@LoginTime
try
{
    Cmd.ExecuteNonQuery();           // 将添加记录
    Response.Write("<script>alert('恭喜你，你已注册成功！')</script>");
    GridView1.Visible = true;         //GridView1 可见
    Panel1.Visible = false;           //Panel1 设置为不可见
    Label1.Text = "已注册的用户";
}
catch (SqlException sqlException)
{ Response.Write("<script>alert('"+ sqlException.Message + "')</script>"); } //显示连接异常信息
finally
{
    if(conn.State == ConnectionState.Open) //判断连接是否是打开状态
        conn.Close();                     //关闭连接
}
}

```

(9) 重输的 Click 事件。双击【重输】按钮。添加代码，如下所示：

```

protected void Button2_Click(object sender, EventArgs e)
{
    //清空所有的 TextBox 控件
    TextBox1.Text = null;
    TextBox2.Text = null;
    TextBox3.Text = null;
    TextBox4.Text = null;
    TextBox5.Text = null;
}

```

(10) 按【Ctrl+F5】组合键运行此网站，输入信息，单击【注册】按钮，结果如图 T7.4 所示。



图 T7.4 注册成功后的页面

#### 4. 思考与练习

- (1) 添加找回密码功能。
- (2) 添加管理员页面，删除用户功能。

# 实验 8 文件系统访问

## 1. 实验目的

- (1) 掌握文件夹的创建和删除操作。
- (2) 掌握在 TreeView 控件中动态加载节点。

## 2. 实验内容

通过 TreeView 控件创建和删除文件夹。

## 3. 实验步骤

设计步骤如下：

- (1) 新建站点。打开 VS 2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，创建一个使用文件系统的 ASP.NET 网站，文件存放的位置是“D:\Exper8”（Exper8 文件夹是已经创建过的），单击【确定】按钮。
- (2) 设计页面。打开 Default.aspx 页面的设计视图，从工具箱的标准栏中选择 1 个 TreeView 控件、2 个 Button 控件和 1 个 Panel 控件放到页面上，在 Panel 控件中放入 1 个 TextBox 控件和 1 个 Button 控件并且输入提示“请输入文件夹名称：”。
- (3) 控件属性设置。Panel1 的 Visible 设置为“False”，Button1、Button2 和 Button3 的 Text 设置为“创建文件夹”、“删除文件夹”和“确定”，如图 T8.1 所示。

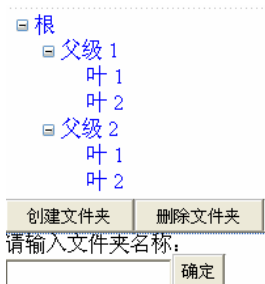


图 T8.1 页面设计部分截图

- (4) 新建文件夹“Files”。打开“解决方案资源管理器”→右击站点→选择“新建文件夹”，命名为“Files”。
- (5) Files 文件夹作为 TreeView1 的根节点，Files 文件夹中的文件夹作为 TreeView1 的子节点。打开 Default.aspx.cs 代码页面，添加命名空间“using System.IO;”。实现代码如下所示：

```
protected string root = "Files"; //根节点的名称
protected void Page_Load(object sender, EventArgs e)
{
```

```

        if (!IsPostBack)                                     //首次加载页面时执行下面代码
        {
            string strCurrentDir = Server.MapPath("./")+root; //获取指定文件夹的实际路径
            ShowDirectory(strCurrentDir);                     //显示所有的文件夹
        }
    }

    private void ShowDirectory(string path)
    {
        TreeView1.Nodes.Clear();
        TreeNode node = new TreeNode(root, Server.MapPath("./")+root); //创建文件夹列表的根节点
        TreeView1.Nodes.Add(node);
        node.Selected = true;                                       //设置根节点的文件夹为默认文件夹
        ShowSubDirectory(node, node.Value);                       //创建指定文件夹的子文件夹节点
    }
    private void ShowSubDirectory(TreeNode parent, string path) //加载子节点
    {
        DirectoryInfo directory = new DirectoryInfo(path);
        foreach (DirectoryInfo d in directory.GetDirectories())
        {
            TreeNode node = new TreeNode(d.Name, d.FullName);
            parent.ChildNodes.Add(node);
            ShowSubDirectory(node, d.FullName);                   //对当前文件夹递归调用本函数
        }
    }
}

```

(6) 选择节点时，让被选择的节点区别于其他的节点，让此节点变色并有提示。切换到设计视图，双击“TreeView1”，添加代码，如下所示：

```

protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e)
{
    TreeView1.SelectedNode.ToolTip = "被选择";                //提示被选择
    TreeView1.SelectedNode.Style.ForeColor = System.Drawing.Color.SkyBlue; //被选择的节点改变颜色
}

```

(7) Button1 的 Click 事件。显示创建文件夹的 Panel1 和里面的控件。双击“Button3”，添加代码，如下所示：

```

protected void Button1_Click(object sender, EventArgs e)
{
    Panel1.Visible = true;
}

```

(8) Button2 的 Click 事件。删除文件夹，双击“Button2”，添加代码，如下所示：

```

protected void Button2_Click(object sender, EventArgs e)
{
    if (TreeView1.Nodes[0].Checked == true)                   //不允许删除根目录
    {

```



```

        Response.Write("<script>alert('不允许删除根目录')</script>");
        return;
    }
    DirectoryInfo dir = new DirectoryInfo(TreeView1.SelectedNode.Name);
    try
    {
        dir.Delete(); //删除文件夹
        ShowDirectory(root); //重新显示所有的文件夹
    }
    catch (Exception)
    { Response.Write("<script>window.alert('文件夹不为空')</script>"); } //不允许删除非空文件夹
}

```

(9) Button3 的 Click 事件。新建文件夹，双击“Button3”，添加代码，如下所示：

```

protected void Button3_Click(object sender, EventArgs e)
{
    if (TextBox1.Text == string.Empty)
        Response.Write("<script>window.alert('请输入要创建的文件夹名称')</script>");
    else
    {
        string strPathName = TextBox1.Text;
        string strFullPath = TreeView1.SelectedNode.Name;
        string strPathFullName = strFullPath + "\\\" + strPathName; //获取新文件夹的实际路径
        Directory.CreateDirectory(strPathFullName); //创建新文件夹
        TreeNode node = new TreeNode(strPathName, strPathFullName);
        TreeView1.SelectedNode.ChildNodes.Add(node); //向文件夹列表中添加新节点
        node.Selected = true; //选中新节点
        TextBox1.Text = string.Empty; //清空 TextBox1
    }
    Panel1.Visible = false; //隐藏创建文件夹的控件
}

```

(10) 按【Ctrl+F5】组合键运行此网站，添加文件夹，结果如图 T8.2 所示。打开“解决方案资源管理器”，Files 中所添加的文件夹如图 T8.3 所示。

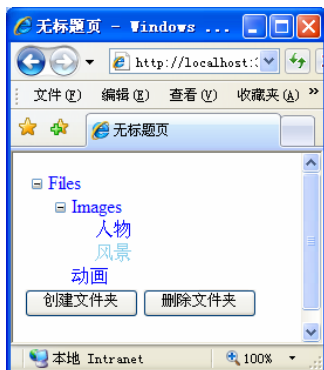


图 T8.2 网站运行的结果



图 T8.3 Files 中所添加的文件夹

#### 4. 思考与练习

(1) 添加上传文件到新建的文件夹中的功能。

(2) 添加一个 GridView 控件，当选择文件夹时，在此 GridView 控件中显示此文件夹所有的文件信息。

# 实验 9 ASP.NET XML 编程

## 1. 实验目的

- (1) 掌握 XML 语言。
- (2) 掌握 ASP.NET 对 XML 的读写。

## 2. 实验内容

设计注册页面，将用户的注册信息保存到 XML 文件中。当然首先在 XML 文件中检测用户名是否存在，如果不存在并且注册码正确，则将用户信息保存到 XML 文件中。

## 3. 实验步骤

设计步骤如下：

(1) 新建站点。打开 VS 2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，创建一个使用文件系统的 ASP.NET 网站，文件存放的位置是“D:\Exper9”（Exper9 文件夹是已经创建过的），单击【确定】按钮。

(2) 设计页面。打开 Default.aspx 页面的设计视图，从工具箱的标准栏中选择 1 个 Label 控件放到页面上。单击菜单“表”→“插入表”，插入 7 行 2 列的表，在表中拖放 6 个 TextBox 控件、1 个 Label 控件和 3 个 Button 控件，并且从工具箱的验证栏中拖放 2 个 RequiredFieldValidator、1 个 CompareValidator、1 个 RangeValidator 和 1 个 RegularExpressionValidator 验证控件，如图 T9.1 所示。

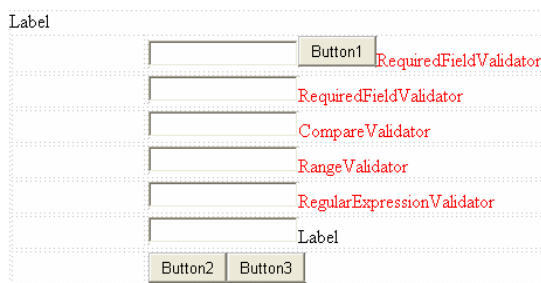


图 T9.1 拖放控件后的页面

(3) 控件属性设置。控件的属性设置如表 T9.1 所示。其中 TextBox2 和 TextBox3 控件的 TextMode 设置为“Password”。Button1、Button2 和 Button3 控件的 Text 设置为“是否可用”、“确定”和“关闭页面”。在表格的适当地方输入提示字样，如图 T9.2 所示。

表 T9.1 控件的属性设置

控件 ID	属 性 设 置
Label1	Size: X-Large; Bold: True
RegularExpressionValidator1	ErrorMessage: Email 格式不对; ControlToCompare: TextBox5; ValidationExpression: Internet 电子邮件地址
RequiredFieldValidator1	ErrorMessage: 用户名不可为空; ControlToValidate: TextBox1
RequiredFieldValidator2	ErrorMessage: 密码不可为空; ControlToValidate: TextBox2
CompareValidator1	ErrorMessage: 密码不一致; ControlToCompare: TextBox2; ControlToValidate: TextBox3
RangeValidator1	ErrorMessage: 请输入 18~60 之间数; ControlToCompare: TextBox4; MaximumValue: 60; MinimumValue: 18

用户注册

用户名:

是否可用

用户名不可为空

密码:

密码不可为空

确认密码:

密码不一致

年龄:

请输入18-60之间数

Email:

Email格式不正确

输入验证码:

Label

确定

关闭页面

图 T9.2 设计后的页面

(4) 创建 XML 文件。打开“解决方案资源管理器”→右击站点→选择“添加新项”→在“添加新项”对话框中选择“XML 文件”→命名为“RegisterInfo.xml”。打开 RegisterInfo.xml 文件，添加节点，代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<members></members>                                //添加节点 members
```

(5) 加载页面生成注册码。打开 Default.aspx.cs 代码页，在 Page\_Load 方法体内添加代码，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)                                //如果第一次加载页面则执行下面代码
    {
        string Vchar = "0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z";
        string[] VcArray = Vchar.Split(new Char[] { ',' }); //把字符串以“,”分割存入数组中
        string VNum = null;                                //定义存放随机数字字符串
        Random rand = new Random();                        //随机数生成器
        for (int i = 1; i < 5; i++)                          //产生 4 个字符串
        {
            int t = rand.Next(35);                          //产生 0~35 之间的随机数
            VNum += VcArray[t];
        }
    }
}
```

```

        Label2.Text = VNum;           //显示随机数
    }
}

```

(6) 检查用户名是否可用。涉及到 XML 文件的操作，添加命名空间 “using System.IO;” 和 “using System.Xml;”。切换到设计视图，双击【是否可用】按钮，添加检查用户名是否可用代码，如下所示：

```

protected static bool isOK = false;           //是否可以存入信息
protected static bool isValid = false;       //用户名是否检查过重复
protected bool Exi_Name()                   //检查 XML 文件中是否有用户名冲突
{
    XmlElement thePerson = null, root = null;
    XmlDocument xmldoc = new XmlDocument();
    xmldoc.Load(Server.MapPath("RegisterInfo.xml"));
    root = xmldoc.DocumentElement;
    //查找指定元素的字符串
    String strTemp = "/members/Person[username='" + TextBox1.Text + "']";
    thePerson = (XmlElement)root.SelectSingleNode(strTemp);
    if (thePerson != null)                   //如果有用户名冲突则返回 false
    {
        return false;
    }
    else                                     //如果没有用户名冲突则返回 true
    {
        return true;
    }
}

```

```

protected void Button1_Click(object sender, EventArgs e)
{

```

```

    if(Exi_Name())                         //调用 Exi_Name 方法判断是否有用户名冲突
    {
        isValid = true;
        isOK = true;
        Response.Write("<script>alert('该名可以使用！')</script>");
    }
    else
    {
        Response.Write("<script>alert('该名已被使用！')</script>");
        return;
    }
}

```

(7) Button2 的 Click 事件。切换到设计视图，双击【确定】按钮，添加注册代码，如下所示：

```

protected void Button2_Click(object sender, EventArgs e)
{
    if (TextBox6.Text.Trim() != Label2.Text.Trim()) //判断注册号是否正确
    {
        Response.Write("<script>alert('验证码输入错误！')</script>");
    }
}

```

```

        return;
    }
    //每个用户信息存在 Person 元素中，用户名、密码、年龄、邮箱均为 Person 的子元素
    try
    {
        //-----用户信息父元素-----子元素-----文档根元素-----
        XmlElement thePerson = null, theElem = null, root = null;
        XmlDocument xmldoc = new XmlDocument();
        xmldoc.Load(Server.MapPath("RegisterInfo.xml"));
        root = xmldoc.DocumentElement;           //获取文档的根
        if (!IsValid)                             //如果没有检查用户名是否可用则检查用户名是否可用
        {
            if (Exi_Name())
            {
                isOK = true;
            }
            else
            {
                Response.Write("<script>alert('该用户名已被使用！')</script>");
                return;
            }
        }
        if (isOK)                                //用户名可用情况下执行下面代码
        {
            thePerson = xmldoc.CreateElement("Person");
            theElem = xmldoc.CreateElement("username");
            theElem.InnerText = TextBox1.Text.Trim();
            thePerson.AppendChild(theElem);        //theElem 作为 thePerson 的子元素
            theElem = xmldoc.CreateElement("password");
            theElem.InnerText = TextBox2.Text.Trim();
            thePerson.AppendChild(theElem);
            theElem = xmldoc.CreateElement("age");
            theElem.InnerText = TextBox4.Text.Trim();
            thePerson.AppendChild(theElem);
            theElem = xmldoc.CreateElement("Email");
            theElem.InnerText = TextBox5.Text.Trim();
            thePerson.AppendChild(theElem);
            root.AppendChild(thePerson);           //thePerson 作为 root 的子元素
            xmldoc.Save(Server.MapPath("RegisterInfo.xml")); //保存信息
        }
    }
    catch (Exception ex)
    {
        Response.Write("<script>alert('"+ex.Message+"')</script>");
    }
    Response.Write("<script>alert('注册成功！')</script>");
    TextBox1.Text = TextBox2.Text = TextBox3.Text = TextBox4.Text
        = TextBox5.Text = TextBox6.Text = null;    //清空所有的 TextBox 控件
}

```

(8) Button3 的 Click 事件。切换到设计视图，双击【关闭页面】按钮，添加关闭页面代

码，如下所示：

```
protected void Button3_Click(object sender, EventArgs e)
{
    Response.Write("<script>window.opener=null;window.close();</script>");//弹出关闭页面对话框
}
```

（9）按【Ctrl+F5】组合键运行网站，见图 T9.3，输入信息，单击【确定】按钮成功后，打开 RegisterInfo.xml 文件，信息存入 XML 文件中。



图 T9.3 页面运行效果



图 T9.4 信息存入 RegisterInfo.xml 文件中

4. 思考与练习

- （1）添加修改用户信息功能。
- （2）添加显示所有的注册用户功能。

## 实验 10 Web服务设计

### 1. 实验目的

- (1) 熟练掌握 ASP.NET Web 服务的创建方法。
- (2) 熟练掌握 ASP.NET 应用程序调用 Web 服务的方法。

### 2. 实验内容

使用 Web 服务实现简单计算器。

### 3. 实验步骤

设计步骤如下：

(1) 新建站点。打开 VS2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，创建一个使用文件系统的 ASP.NET 网站，文件存放的位置是“D:\Exper10”（Exper10 文件夹是已经创建过的），单击【确定】按钮。

(2) 新建 Web 服务。打开“解决方案资源管理器”→右击站点→选择“添加新项”，在弹出的“添加新项”对话框中选择“Web 服务”模板，命名为“WebService.asmx”，单击【确定】按钮。

(3) 为 Web 服务添加代码。打开 WebService.cs 代码页，添加代码，如下所示：

```
... //前面部分在这用“...”代替
public class WebService : System.Web.Services.WebService
{
    public WebService () {
        //如果使用设计的组件，请取消注释以下行
        //InitializeComponent();
    }
    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }
    [WebMethod]
    public double Sum(double a, double b) //计算两个数的和
    { return a + b; }
    [WebMethod]
    public double Sub(double a, double b) //计算两个数的差
    { return a - b; }
    [WebMethod]
    public double Mult(double a, double b) //计算两个数的积
}
```



```

    { return a * b; }
    [WebMethod]
    public double Div(double a, double b)    //计算两个数的商
    { return a / b; }
}

```

(4) 添加 Web 引用。打开“解决方案资源管理器”→右击站点→选择“添加 Web 引用”，在“添加 Web 引用”对话框中选择“此解决方案中的 Web 服务”，如图 T10.1 所示，选择“WebService”服务，如图 T10.2 所示，单击【添加引用】按钮。

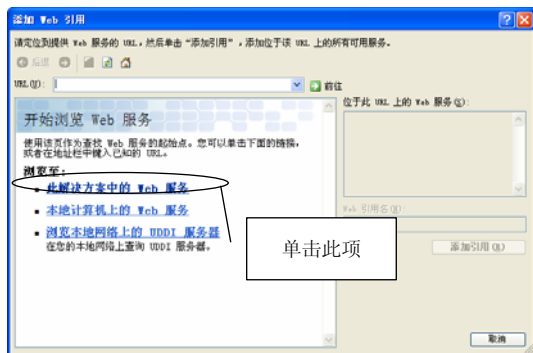


图 T10.1 选择此解决方案中的 Web 服务

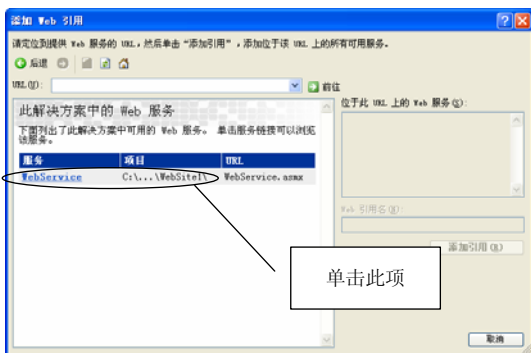


图 T10.2 选择“WebService”服务

(5) 新建 Calculator.aspx 页面。打开“解决方案资源管理器”→右击站点→选择“添加新项”，在弹出的“添加新项”对话框中选择“Web 窗体”模板，命名为“Calculator.aspx”，单击【确定】按钮。

(6) 设计 Calculator.aspx 页面。切换到 Calculator.aspx 页面的设计视图，从工具箱中拖放 1 个 TextBox 控件和 1 个 Button 控件。右击 Button1，复制 15 个 Button 控件，控件的布局如图 T10.3 所示，按图修改各个 Button 控件的 Text 属性，其中 ID 为“Button16”的 Text 属性设置为“=”。TextBox 的 ReadOnly 属性设置为“True”。

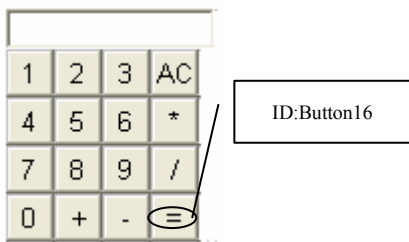


图 T10.3 页面设计

(7) 计算按钮【=】的 Click 事件。双击【=】按钮，添加代码，如下所示：

```


protected void Button16_Click(object sender, EventArgs e)    //单击“=”，计算
{
    if (this.TextBox1.Text == "" && temp1.ToString() != null)
    {

```

```

        this.TextBox1.Text = temp1.ToString();
        dot = true;
    }
    else
    {
        double temp3 = 0.0;
        temp2 = Convert.ToDouble(this.TextBox1.Text);
        localhost.WebService result = new localhost.WebService();
        switch (m)
        {
            case 0:
                temp3 = result.Sum(temp1, temp2);           //+
                break;
            case 1:
                temp3 = result.Sub(temp1, temp2);           //-
                break;
            case 2:
                temp3 = result.Div(temp1, temp2);           ///
                break;
            case 3:
                temp3 = result.Mult(temp1, temp2);          //*
                break;
            default:
                Response.Write("数据有误，请重新输入！");
                break;
        }
        if (temp3 > double.MaxValue)
        {
            Response.Write("<script>alert('结果值超出双精度最大值！')</script>");
            return;
        }
        this.TextBox1.Text = temp3.ToString(); //显示计算结果
        dot = true;
    }
}

```

（8）添加数据按钮和算术符号按钮事件代码。首先添加代码，代码如下所示。然后切换到设计视图，分别打开数据按钮和算术符号按钮的属性窗口，单击  图标，选择“number\_Click”方法，算术符号按钮选择“Cal\_Click”方法。

```

public static double temp1;           //存储第一个变量
public static double temp2;           //存储第二个变量
public static int m;                   //保存输入的状态
public bool dot = false;               //判断是否单击“=”号
protected void number_Click(object sender, EventArgs e) //数据按钮
{
    Button btn_select = (Button)sender; //拆箱操作

```

```

        if (dot != true)
        {
            this.TextBox1.Text += btn_select.Text;           //显示单击按钮所输入的字符
        }
    }
    protected void Cal_Click(object sender, EventArgs e)       //算术符号按钮
    {
        if (this.TextBox1.Text != "")
        {
            temp1 = Convert.ToInt32(this.TextBox1.Text);
            this.TextBox1.Text = "";
            Button btn_select = (Button)sender;               //拆箱操作
            switch (btn_select.Text)
            {
                case "+":                                       //如果单击+号则 m=0
                    m=0;
                    break;
                case "-":                                       //如果单击-号则 m=1
                    m=1;
                    break;
                case "/":                                       //如果单击/号则 m=2
                    m=2;
                    break;
                case "*":                                       //如果单击*号则 m=3
                    m=3;
                    break;
            }
        }
    }
}

```

(9) 清空【AC】按钮事件。切换到设计视图，双击【AC】按钮，添加代码，如下所示：

```

protected void Button15_Click(object sender, EventArgs e)//单击 AC，清空
{
    dot = false;
    this.TextBox1.Text = "";
    temp1 = 0.0;
    temp2 = 0.0;
}

```

同时在 Page\_Load 方法体内添加标题代码，如下所示：

```

protected void Page_Load(object sender, EventArgs e)
{
    this.Title = "调用 Web 服务实现简单的计算器";
}

```

(10) 运行网站。按【Ctrl+F5】组合键运行此网站，计算一下数据，结果如图 T10.4 所示。

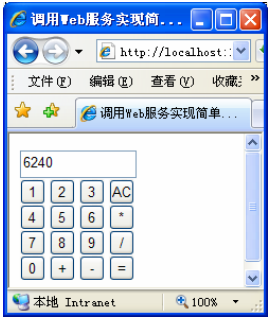


图 T10.4 调用 Web 服务计算

4. 思考与练习

- (1) 设计一个数据库并通过 Web 服务的方式查找数据库信息。
- (2) 通过 Web 服务的方式向数据库中添加信息。

# 实验 11    AJAX应用

## 1. 实验目的

- (1) 掌握 ASP.NET AJAX 编程的基本方法。
- (2) 掌握 ScriptManager、UpdatePanel、Timer 基本控件的使用方法。

## 2. 实验内容


通过 Timer 控件不停改变 Label 控件的字体大小。同时为了体现 AJAX 的应用，另加一个 Lable 控件，字体时刻不变，以体现区别。

## 3. 实验步骤

设计步骤如下所示：

(1) 新建站点。打开 VS 2008，单击菜单栏“文件”→“新建”→“网站”，弹出“新建网站”对话框，选择“ASP.NET 网站”模板，创建一个使用文件系统的 ASP.NET 网站，文件存放的位置是“D:\Exper11”（Exper11 文件夹是已经创建过的），单击【确定】按钮。

(2) 设计页面。打开 Default.aspx 页面的设计视图，从工具箱的标准栏中拖放 1 个 Label 控件到页面上，此控件的 Text 属性设置为“不变”；从工具箱的 AJAX Extensions 栏中拖放 1 个 ScriptManager 控件和 1 个 UpdatePanel 控件到页面上，再拖放 1 个 Label 控件和 1 个工具箱的 AJAX Extensions 栏的 Timer 控件到 UpdatePanel 控件中，此 Label 的 Text 属性设置为“变大”，如图 T11.1 所示。

(3) Timer1 的属性设置。打开 Timer1 的属性窗口，Interval 属性设置为“1000”，单击 图标，双击 Tick 后的空白处，系统自动添加了“Timer1\_Tick”方法。

(4) 添加代码。代码如下所示：

```
protected static int a = 8;
protected void Timer1_Tick(object sender, EventArgs e)           //Timer1 的 Tick 事件代码
{
    a+=5;                                                         //Timer1 每触发一次 a+5
    Label1.Font.Size = a;
    Label2.Font.Size = a;                                         //改变 Label2 字体大小
}
```

(5) 运行网站。按【Ctrl+F5】组合键运行网站。“变大”字样不停变大，而“不变”字样却不变，如图 T11.2 所示。

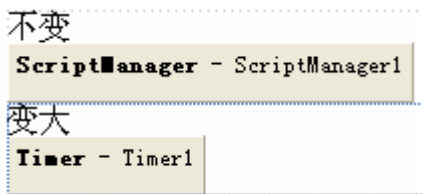


图 T11.1 页面设计

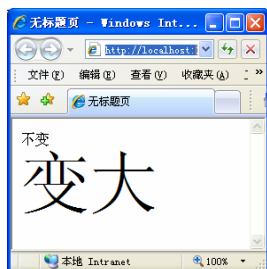


图 T11.2 网页运行的效果

#### 4. 思考与练习

- (1) 试从 <http://www.asp.net/AJAX> 网站链接下载 ASP.NET AJAX 控件工具集。
- (2) 将下载的 AJAX 控件工具集添加到 VS 2008 工具箱中。

**提示 :**在工具箱中新建一个选项卡,然后右击此选项卡,在弹出的菜单中选择“选择项”,在“选择工具箱项”窗口中单击【浏览】按钮,选择下载的 Bin 文件夹中的 AjaxControlToolkit.dll 文件,然后单击【确定】按钮即可。

# 第三部分 实 习

## 综合应用实例：BBS系统

本实习运用前面学习的知识，以一个小型的 ASP.NET 学习论坛（BBS 系统）为实例，来介绍如何开发 ASP.NET 应用程序。

### P.1 系统功能设计

通常的 BBS 系统的功能要素包括用户注册、用户登录、帖子查询、发帖、回帖和管理员维护等。根据这些要素可以设计出的小型 ASP.NET 学习论坛的功能模块如下。

- (1) 用户注册：用户名不允许重名。
- (2) 用户登录：允许注册用户和访客登录。
- (3) 查询主帖：分页显示主帖的标题等信息。
- (4) 详细信息：查询主帖的详细信息及其全部回复信息。
- (5) 发表新主帖：可以输入新帖并插入数据库中。
- (6) 回复：对某个主帖进行回复。
- (7) 管理员登录：只允许管理员登录。
- (8) 删除主帖及其全部回复：可以在主帖列表中选择并删除某个主帖。

### P.2 系统流程设计

由于本系统功能比较简单，所以各功能模块都可以用一个独立的页面实现。图 P.1 展示了本系统各功能模块之间的流程结构。

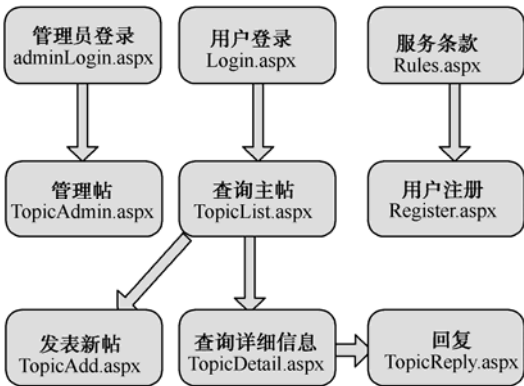


图 P.1 系统流程图

在本例的系统流程中，普通用户和游客通过首页登录，可以进入主帖查询页面查看全部主帖标题列表，通过其中的链接可以进入发表新帖页面实现发帖。另外，用户可以通过“详细信息”链接进入查看详细信息的页面，在详细信息页面中可以进行回复操作。而管理员需要通过单独的登录页面来验证身份，然后在查询主帖页面中进行帖子的删除和维护管理。如果要注册用户，则可以进入接受服务条款页面进入用户注册页面注册新用户。

### P.3 数据库设计

根据前面介绍的功能需求，下面设计本系统的数据库 MyBBS.mdf，包括用户表（User）、主帖表（Topic）、回帖表（Reply）和管理员信息表（adminUser）。

设计步骤如下：

- （1）新建网站。在 D 盘新建文件夹“Practice”，打开 VS 2008→单击菜单“文件”→“新建”→“网站”，在弹出的“新建网站”对话框中选择“ASP.NET 网站”模板，保存位置为文件系统的“D:\Practice”文件夹，单击【确定】按钮完成网站的新建。
- （2）新建数据库。打开“解决方案资源管理器”→右击“App\_Data”文件夹→选择“添加新项”→选择“SQL Server 数据库”模板→命名为“MyBBS.mdf”→单击【添加】按钮。
- （3）添加表。打开“服务器资源管理器”→右击 MyBBS.mdf 数据库下的“表”→选择“添加新表”选项，如图 P.2 所示。在弹出的表的结构设计窗口中设计表的结构，根据表 P.1 设计用户表（User），设计后如图 P.3 所示，关闭窗口保存为“User”。

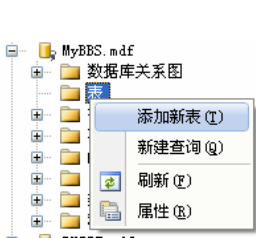


图 P.2 添加新表

列名	数据类型	允许 Null
UserID	int	<input type="checkbox"/>
LoginName	varchar (50)	<input type="checkbox"/>
UserName	varchar (50)	<input type="checkbox"/>
Password	varchar (50)	<input type="checkbox"/>
Address	varchar (100)	<input checked="" type="checkbox"/>
Homepage	varchar (50)	<input checked="" type="checkbox"/>
Email	varchar (50)	<input checked="" type="checkbox"/>

图 P.3 设计用户表

其中列属性设置如下。

- ① 设置 UserID 为主键。右击“UserID”前处，选择“设置主键”。
- ② 设置 UserID 为自动加 1。选择“UserID”，在列属性窗口中展开“标识规范”，单击“(是标识)”，选择“是”，如图 P.4 所示。



图 P.4 设置自动增量

表 P.1 是用于存储注册用户信息的数据表 User，其主键是 UserID，LoginName 字段是用户



登录名，在登录页中使用该名称来登录，而 `UserName` 是用户注册的全名，不是登录用的名称。

表 P.1 用户表 ( `User` )

字 段 名	字 段 类 型	允 许 空	说 明
<code>UserID</code>	<code>int</code>	否	用户唯一标识，主键，自动增量
<code>LoginName</code>	<code>varchar(50)</code>	否	登录名
<code>UserName</code>	<code>varchar(50)</code>	否	用户名
<code>Password</code>	<code>varchar(50)</code>	否	密码
<code>Address</code>	<code>varchar(100)</code>	是	住址
<code>Homepage</code>	<code>varchar(50)</code>	是	个人主页
<code>Email</code>	<code>varchar(50)</code>	是	邮箱地址

根据其他的表格设计数据库的表：

表 `P.2` 是用于存储用户发布的主帖信息的数据表 `Topic`，其主键是 `TopicID`，字段 `UserLoginName` 是发帖用户的登录名，`Title` 是帖子的标题，`Content` 是主帖的详细内容。

表 P.2 主帖表 ( `Topic` )

字 段 名	字 段 类 型	允 许 空	说 明
<code>TopicID</code>	<code>int</code>	否	主帖唯一标识，主键，自动增量
<code>UserLoginName</code>	<code>varchar(50)</code>	否	发帖者登录名
<code>Title</code>	<code>varchar(50)</code>	否	主帖标题
<code>Content</code>	<code>varchar(50)</code>	否	主帖内容
<code>CreateTime</code>	<code>varchar(100)</code>	是	发帖时间
<code>IP</code>	<code>varchar(50)</code>	是	用户机器 IP

表 `P.3` 是用于存储用户回复帖子信息的数据表 `Reply`，其主键是 `ReplyID`，`TopicID` 字段与主帖表的 `TopicID` 字段关联，`UserLoginName` 是回帖用户的登录名，`Title` 是回帖的标题，`Content` 是回帖的详细内容。

表 P.3 回帖表 ( `Reply` )

字 段 名	字 段 类 型	允 许 空	说 明
<code>ReplyID</code>	<code>int</code>	否	回帖唯一标识，主键，自动增量
<code>TopicID</code>	<code>int</code>	否	主帖标识，与主帖关联
<code>UserLoginName</code>	<code>varchar(50)</code>	否	发帖者登录名
<code>Title</code>	<code>varchar(50)</code>	否	回帖标题
<code>Content</code>	<code>varchar(50)</code>	否	回帖内容
<code>CreateTime</code>	<code>varchar(100)</code>	是	发表时间
<code>IP</code>	<code>varchar(50)</code>	是	用户机器 IP

表 P.4 是用于存储注册管理员信息的数据表 `adminUser`，其主键是 `UserID`，`LoginName` 字段是管理员登录名，在登录页中使用该名称来登录，而 `UserName` 是用户注册的全名，不是登录用的名称。

表 P.4 管理员信息表 ( `adminUser` )

字 段 名	字 段 类 型	允 许 空	说 明
UserID	int	否	用户唯一标识，主键，自动增量
LoginName	varchar(50)	否	管理员登录名
UserName	varchar(50)	否	用户名
Password	varchar(50)	否	密码

所有的表设计完成后如图 P.5 所示。

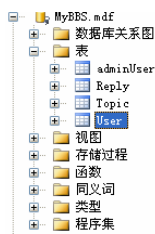


图 P.5 设计完成后的数据库

(4) 添加确发器。当删除了主帖表 (`Topic`) 中的帖子时，同时也要删除回帖表 (`Reply`) 中此帖的回复的帖子。打开“服务器资源管理器”→右击“`Topic`”表→选择“添加新确发器”，在添加确发器窗口中用下面代码覆盖系统自动生成的代码：

```
ALTER TRIGGER Trigger_Delete
ON dbo.Topic
FOR DELETE
AS
BEGIN
    DELETE FROM Reply WHERE TopicID IN (SELECT TopicID FROM DELETED)
END
```

## P.4 母版页设计

本系统仅使用了一个母版页 `MasterPage.master` 来统一页面的风格。由于本例是小型的 BBS 系统，网站页面不多，纵深不深，访问者迷路的可能性较小，因此在母版页中只使用了简单的导航菜单 `SiteMapPath`。设计后的母版页如图 P.6 所示。



图 P.6 设计后的母版页部分截图

## P.4.1 添加站点地图

设计步骤如下所示：

(1) 添加站点地图。打开“解决方案资源管理器”→右击站点→选择“添加新项”→选择“站点地图”→命名为“Web.sitemap”→单击【添加】按钮。

(2) 添加代码。在 Web.sitemap 中添加代码，如下所示（阴影部分是读者要添加的）：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
<siteMapNode title="用户登录" url="~/Login.aspx" description="登录">
  <siteMapNode title="主帖列表" url="~/TopicList.aspx" description="主帖列表">
    <siteMapNode title="详细信息" url="TopicDetail.aspx" description="详细信息">
      <siteMapNode url="TopicReply.aspx" title="回复" />
    </siteMapNode>
    <siteMapNode url="TopicAdd.aspx" title="发新帖" description="发新帖">
    </siteMapNode>
  </siteMapNode>
</siteMapNode>
</siteMap>
```

## P.4.2 添加母版页

设计步骤如下所示：

(1) 添加母版页。打开“解决方案资源管理器”→右击站点→选择“添加新项”→选择“母版页”→命名为“MasterPage.master”→单击【添加】按钮。

(2) 添加图片。打开“解决方案资源管理器”→右击站点→选择“新建文件夹”→命名为“images”。右击 images 文件夹→选择“添加现有项”→选择准备好的图片→单击【确定】按钮。

(3) 添加表。切换到设计视图，单击菜单“表”→“插入表”，添加 1 个 2 行 2 列的表。

(4) 添加控件。在表格的第 1 行第 1 列中添加 1 个 Calendar 控件，在第 2 列中添加 1 个 Image 控件和 1 个 SiteMapPath 控件。在第 2 行的第 1 列中添加 3 个 LinkButton 控件，将 ContentPlaceHolder 控件发到第 2 列中。

(5) 添加内容页。打开“解决方案资源管理器”→右击站点→选择“添加新项”→选择“Web 窗体”，在“添加新项”对话框中，命名为“adminLogin.aspx”，选中“选择母版页”复选框，单击【添加】按钮→选择“MasterPage.master”→单击【确定】按钮。依照同样的方法分别添加“Login.aspx”、“Register.aspx”、“Rules.aspx”、“TopicAdd.aspx”、“TopicAdmin.aspx”、“TopicDetail.aspx”、“TopicList.aspx”和“TopicReply.aspx”内容页。

(6) 控件的属性设置。Image 控件的 ImageUrl 选择为“~/images/asptop.JPG”。3 个

LinkButton 的 Text 属性分别设置为“注册”、“用户登录”和“管理员登录”，PostBackUrl 分别选择为“~/Rules.aspx”、“~/Login.aspx”和“~/TopicAdmin.aspx”。分别链接到注册、用户登录和管理员登录页面。

## P.5 主题设计

应用主题功能可以轻松地实现对网站美观的控制。可以将样式和布局信息分解为单独的文件组。

设计步骤如下所示：

(1) 添加主题。打开“解决方案资源管理器”→右击站点→“添加 ASP.NET 文件夹”→“主题”，系统会自动创建 App\_Themes 目录，并在其下创建“主题 1”目录。

(2) 添加外观文件和样式表文件。右击“主题 1”→选择“添加新项”，在弹出的窗口中选择“外观文件”，用同样的方法在主题内添加一个样式表文件。使用默认名称。

(3) 设置外观文件。打开 Default.aspx 设计页面，在页面上拖放 2 个 Label 控件和 1 个 TextBox 控件。按表 P.5 所示分别设置 Label 控件和 TextBox 控件的外观属性。打开源视图，将 XHTML 代码粘贴到外观文件 SkinFile.skin 中，并修改代码，修改后的代码如下所示：

```
<asp:Label SkinID="skinLabel1" runat="server" Font-Bold="True" ForeColor="Blue"></asp:Label>
<asp:Label SkinID="skinLabel2" runat="server" Font-Bold="True" Font-Size="X-Large"
    ForeColor="Blue"></asp:Label>
<asp:TextBox SkinID="skinTextBox1" runat="server" BackColor="#CCCCFF"></asp:TextBox>
```

表 P.5 控件属性的设置

控件 ID	属 性 设 置
Label1	Font-Bold="True" , ForeColor="Blue"
Label2	Font-Bold="True" , Font-Size="XX-Large" , ForeColor="Blue"
TextBox1	BackColor="#CCCCFF"

(4) 设置样式表文件。打开样式表文件 StyleSheet.css，右击 body 的“{}”，选择“生成样式”，在弹出的“修改样式”对话框中选择“背景”，background-color 设置为“#99CCFF”。

(5) 打开各个内容页的属性窗口，选择“DOCUMENT”，Theme 属性选择为“主题 1”，如图 P.7 所示。

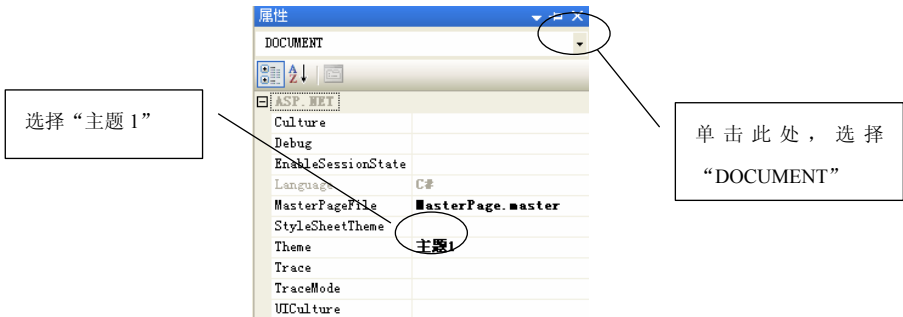


图 P.7 为各个内容页选择主题

## P.6 全局变量

为了便于修改数据库连接，把数据库连接字符串放在配置文件中。  
设计步骤如下所示：

打开 web.config 文件，将<connectionStrings/>修改为下面的代码：

```
<connectionStrings>
  <add name="MyBBSCConnectionString" connectionString="Data Source=
    \SQLEXPRESS;AttachDbFilename=|DataDirectory|\MyBBS.MDF;Integrated Security=
    True;Connect Timeout=30;User Instance=True" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

## P.7 注册模块设计

注册模块包括服务条款页面（Rules.aspx）和用户注册页面（Register.aspx）。当同意了服务条款后链接到“Register.aspx”注册。

### P.7.1 服务条款页面设计

设计步骤如下所示：

- （1）添加服务条款。打开 Rules.aspx 页面设计视图，在“ContentPlaceHolder”的控件中添加服务条款。这里的条款读者可以自己编写添加。
- （2）添加接受和不接受链接。在条款后添加 2 个 LinkButton 控件，Text 属性设置分别为“接受”和“不接受”。PostBackUrl 属性选择为“~/Register.aspx”和“~/Login.aspx”。

### P.7.2 用户注册页面设计

设计步骤如下所示：

- （1）设计页面。打开 Register.aspx 页面的设计视图，插入 9 行 2 列的表，在表中拖放 1 个 Label 控件、7 个 TextBox 控件和 2 个 Button 控件，并且从工具箱的验证栏中拖放 3 个 RequiredFieldValidator、1 个 CompareValidator 和 1 个 RegularExpressionValidator 验证控件。
- （2）控件属性设置。控件的属性设置如表 P.6 所示。其中 TextBox3 和 TextBox4 的 TextMode 设置为“Password”。Button1 和 Button2 的 Text 设置为“提交”和“重输”。在表格的适当地方输入提示字样，如图 P.8 所示。

表 P.6 控件的属性设置

控件 ID	属 性 设 置
Label1	Text: 用户注册, SkinID: skinLabel2
RegularExpressionValidator1	ErrorMessage: 邮箱格式不正确、ControlToCompare: TextBox7、ValidationExpression: Internet



```

Cmd.CommandText = SqlStr;
// 添加参数对象,并给参数赋值
Cmd.Parameters.Add("@LoginName", SqlDbType.VarChar, 50).Value = TextBox1.Text.Trim();
Cmd.Parameters.Add("@UserName", SqlDbType.VarChar, 50).Value = TextBox2.Text.Trim();
Cmd.Parameters.Add("@PassWord", SqlDbType.VarChar, 50).Value = TextBox3.Text.Trim();
Cmd.Parameters.Add("@Address", SqlDbType.VarChar, 100).Value = TextBox5.Text.Trim();
Cmd.Parameters.Add("@Homepage", SqlDbType.VarChar, 50).Value = TextBox6.Text.Trim();
Cmd.Parameters.Add("@Email", SqlDbType.VarChar, 50).Value = TextBox7.Text.Trim();
try
{
    Cmd.ExecuteNonQuery();           // 将添加记录
    Response.Write("<script>alert('恭喜你, 你已注册成功! ')</script>");
}
catch (SqlException sqlException)    // 显示连接异常信息
{
    Response.Write("<script>alert('\" + sqlException.Message + '\"</script>");
    return;
}
finally
{
    if (conn.State == ConnectionState.Open)    //如果数据库连接打开则关闭连接
        conn.Close();
}
}

```

(4) 按钮“重输”的 Click 事件。双击【重输】按钮，添加代码，如下所示：

```

protected void Button2_Click(object sender, EventArgs e)
{
    //清空所有的 TextBox 控件
    TextBox1.Text = null;
    TextBox2.Text = null;
    TextBox3.Text = null;
    TextBox4.Text = null;
    TextBox5.Text = null;
    TextBox6.Text = null;
    TextBox7.Text = null;
}

```

## P.8 登录模块设计

登录模块包括用户登录（Login.aspx，包括游客登录）和管理员登录（adminLogin.aspx）。用户和游客都可以浏览、发表帖，但是游客不能回复帖。管理员可以浏览、回复和删除帖。

### P.8.1 用户登录页面设计

设计步骤如下所示：

(1) 设计页面。打开 Login.aspx 页面的设计视图，插入 4 行 2 列的表，在表中放入 1 个 Label 控件、2 个 TextBox 控件和 2 个 Button 控件。

(2) 控件属性设置。TextBox2 的 TextMode 设置为“Password”。Button1 和 Button2 的 Text 设置为“登录”和“游客”。Label1 的 Text 设置为“用户登录”，SkinID 为“skinLabel2”，在表格的适当地方输入提示字样，如图 P.9 所示。



图 P.9 用户登录页面设计

(3) 按钮“登录”的 Click 事件。双击【登录】按钮，添加代码，首先添加命名空间“using System.Data.SqlClient;”，代码如下所示：

```
protected void Button1_Click(object sender, EventArgs e) //登录
{
    string userLoginName = TextBox1.Text.Trim(); //用户登录名
    string userPassword = TextBox2.Text.Trim(); //密码
    string sqlcon = System.Configuration.
        ConfigurationManager.ConnectionStrings["MyBBSCConnectionString"].ToString();
    string sqlcomtext = "select * from [User] where LoginName =" + userLoginName +
        "and Password=" + userPassword + """;
    SqlConnection conn = new SqlConnection(sqlcon);
    conn.Open(); //打开数据库连接
    // 检查用户是否已存在
    SqlCommand sqlcom = new SqlCommand(sqlcomtext, conn);
    if (sqlcom.ExecuteScalar() != null) //如果存在
    {
        Session.Add("login_name", userLoginName); //使用 Session 来保存用户登录名信息
        conn.Close();
        Response.Redirect("TopicList.aspx"); //重定向到查询主帖页
    }
    else //如果用户不存在
    { Response.Write("<Script>alert('对不起，用户名和密码不一致！')</Script>"); }
    if (conn.State == ConnectionState.Open) //如果数据源是连接状态，则断开连接
    { conn.Close(); }
}
```

(4) 按钮“游客”的 Click 事件。双击【游客】按钮。添加代码，如下所示：

```
protected void Button2_Click(object sender, EventArgs e) //游客
{
    Session.Add("login_name", "guest"); //使用 Session 来保存游客名 guest 信息
    Response.Redirect("TopicList.aspx"); //重定向到查询主帖页
}
```



}

## P.8.2 管理员登录页面设计

设计步骤如下所示：

(1) 设计页面。打开 adminLogin.aspx 页面的设计视图，插入 4 行 2 列的表，在表中放入 1 个 Label 控件、2 个 TextBox 控件和 1 个 Button 控件。

(2) 控件属性设置。TextBox2 的 TextMode 设置为“Password”。Button1 的 Text 设置为“登录”。Label1 的 Text 设置为“管理员登录”，SkinID 为“skinLabel2”，在表格的适当地方输入提示字样，如图 P.10 所示。

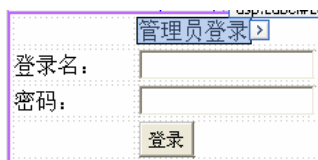


图 P.10 管理员登录页面设计

(3) 按钮“登录”的 Click 事件。双击【登录】按钮，添加代码，首先添加命名空间“using System.Data.SqlClient;”，代码如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    string userLoginName = TextBox1.Text.Trim(); //用户登录名
    string userPassword = TextBox2.Text.Trim(); //密码
    string sqlcon = System.Configuration.
        ConfigurationManager.ConnectionStrings["MyBBSCConnectionString"].ToString();
    string sqlcomtext = "select * from [adminUser] where LoginName='" + userLoginName + "'and
        Password='" + userPassword + "'";
    SqlConnection conn = new SqlConnection(sqlcon);
    conn.Open(); //打开数据库连接
    // 检查用户是否已存在
    SqlCommand sqlcom = new SqlCommand(sqlcomtext, conn);
    if (sqlcom.ExecuteScalar() != null) //如果存在
    {
        Session.Add("admin", userLoginName); //使用 Session 来保存管理员登录名信息
        Session.Add("login_name", userLoginName); //使用 Session 来保存用户登录名信息
        conn.Close();
        Response.Redirect("TopicAdmin.aspx"); //重定向到管理帖页面
    }
    else //如果用户不存在
    { Response.Write("<Script>alert('对不起，用户名和密码不一致！')</Script>"); }
    if (conn.State == ConnectionState.Open) //如果数据源是连接状态，则断开连接
    { conn.Close(); }
}
```

P.9 发帖模块设计

发帖模块中包括查询主贴页面（TopicList.aspx）、发表新帖页面（TopicAdd.aspx）和查看详细信息页面（TopicDetail.aspx）。在查询主贴页面中可以链接到发表新帖页面也可以链接到查看详细信息页面。

P.9.1 查询主贴页面设计

设计后的查询主贴页面如图 P.11 所示。

编号	标题	用户	发表时间	详细内容
0	abc	abc	2009-6-25 0:00:00	<a href="#">详细信息</a>
1	abc	abc	2009-6-25 0:00:00	<a href="#">详细信息</a>
2	abc	abc	2009-6-25 0:00:00	<a href="#">详细信息</a>
3	abc	abc	2009-6-25 0:00:00	<a href="#">详细信息</a>
4	abc	abc	2009-6-25 0:00:00	<a href="#">详细信息</a>
5	abc	abc	2009-6-25 0:00:00	<a href="#">详细信息</a>
6	abc	abc	2009-6-25 0:00:00	<a href="#">详细信息</a>
7	abc	abc	2009-6-25 0:00:00	<a href="#">详细信息</a>
8	abc	abc	2009-6-25 0:00:00	<a href="#">详细信息</a>
9	abc	abc	2009-6-25 0:00:00	<a href="#">详细信息</a>
1 2				
[LabelPages]			<a href="#">发表新帖&gt;&gt;</a>	
SqlDataSource - SqlDataSource1				

图 P.11 TopicList.aspx 设计后的部分截图

设计步骤如下所示：

- （1）设计页面。打开 TopicList.aspx 页面的设计视图，从工具箱中拖放 1 个 GridView 控件和 1 个 SqlDataSource 控件，在 GridView 控件下再放入 1 个 ID 设置为“LabelPage”的 Label 控件和 HyperLink 控件。
- （2）配置数据源。单击 SqlDataSource 控件的“”图标，选择“配置数据源”→在“配置数据源”对话框中选择“MyBBSCConnectionString”→单击【下一步】按钮→在“配置 Select 语句”对话框中选择“Topic”表，选择“TopicID”、“UserLoginName”、“Title”和“CreateTiem”列→单击【下一步】按钮→单击【完成】按钮。
- （3）GridView 控件的设置。单击 GridView 控件的“”图标，在 GridView 任务中选择数据源为“SqlDataSource1”，选择“启用分页”和“启用排序”复选框。单击“编辑列”项，“HeaderText”分别设置为字段对应的汉字。添加 1 个 HyperLinkField，所对应的“HeaderText”设置为“详细内容”，“Text”设置为“详细内容”，“DataNavigateUrlFields”设置为“TopicID”，“DataNavigateUrlFormatString”设置为“TopicDetail.aspx?topic\_id={0}”，表示链接到 TopicDetail.aspx 页并把 TopicId 值传过去，如图 P.12 所示。

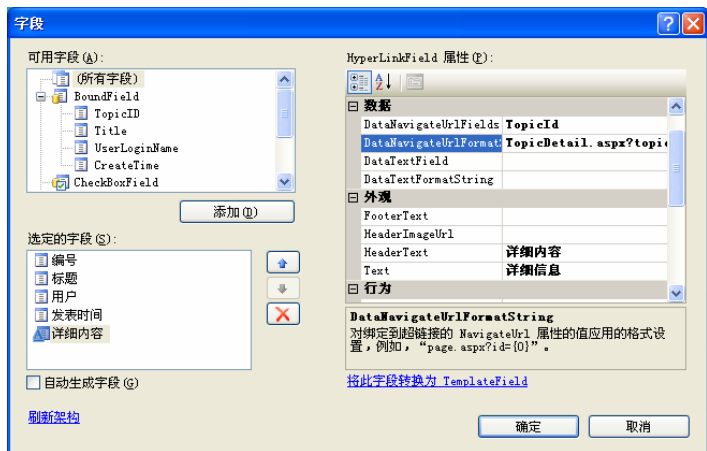


图 P.12 编辑 GridView 中的字段

(4) HyperLink 控件的属性设置。HyperLink 控件的 Text 属性设置为“发表新帖>>”，NavigateUrl 设置为“TopicAdd.aspx”，表示链接到发表新帖页面。

(5) GridView 控件的 PageIndexChanging 事件。打开 GridView 控件的属性窗口，选择“⚡”图标，双击“PageIndexChanging”空白处，添加代码，如下所示：

```
protected void GridView1_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    GridView1.PageIndex = e.NewPageIndex;
    GridView1.DataBind(); //重新绑定
    LabelPages.Text = "查询结果（第" + (GridView1.PageIndex + 1).ToString() + "页 共"
        + (GridView1.PageCount).ToString() + "页）"; //显示所在页数
}
```

(6) 显示所在的页数。在 Page\_Load 方法体内添加代码，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    GridView1.DataBind(); //重新绑定
    LabelPages.Text = "查询结果（第" + (GridView1.PageIndex + 1).ToString() + "页 共" +
        (GridView1.PageCount).ToString() + "页）";
}
```

## P.9.2 发表新帖页面设计

设计后的发表新帖页面如图 P.13 所示。单击【确定】按钮发表新帖，单击【返回】按钮返回到过去的页面。



图 P.13 TopicAdd.aspx 设计后的部分截图

设计步骤如下所示：

(1) 设计页面。打开 TopicAdd.aspx 页面的设计视图，插入 4 行 2 列的表格。在表格的适当位置放入 1 个 Label 控件、2 个 TextBox 控件和 2 个 Button 控件，并输入提示字样。Label 控件的 Text 设置为“发新帖”，SkinID 设置为“skinLabel2”。TextBox1 的 SkinID 设置为“skinTextBox1”。TextBox2 的 SkinID 设置为“skinTextBox1”，TextMode 设置为“MultiLine”，见图 P.12。

(2) 按钮“确定”的 Click 事件。双击【确定】按钮，添加代码，首先添加命名空间“using System.Data.SqlClient;”，代码如下所示：

```
protected void Button1_Click(object sender, EventArgs e)
{
    string constr = ConfigurationManager.ConnectionStrings["MyBBSSConnectionString"].ToString();
    string SqlStr = "Insert into [Topic]([UserLoginName],[Title],[Content],[CreateTime],[IP])
        values(@UserLoginName,@Title,@Content,@CreateTime,@IP)";
    SqlConnection conn = new SqlConnection(constr);
    // 检查用户是否存在
    SqlCommand Cmd = new SqlCommand(SqlStr,conn);
    conn.Open();    //打开数据库连接
    // 添加参数对象,并给参数赋值
    Cmd.Parameters.Add("@UserLoginName", SqlDbType.VarChar, 50).Value = Session["login_name"];
    Cmd.Parameters.Add("@Title", SqlDbType.VarChar,50).Value = TextBox1.Text;
    Cmd.Parameters.Add("@Content", SqlDbType.Text).Value = TextBox2.Text;
    Cmd.Parameters.Add("@CreateTime", SqlDbType.DateTime).Value = DateTime.Now;
    Cmd.Parameters.Add("@IP", SqlDbType.Char, 15).Value = Request.UserHostAddress.ToString();
    try
    {Cmd.ExecuteNonQuery();    }           //添加记录
    catch (SqlException sqlException)    //显示连接异常信息
    {
        Response.Write("<script>alert('"+ sqlException.Message+ "')</script>");
        return;
    }
    finally
    {conn.Close();}           //断开数据库连接
    Button2_Click(null,null);    //调用 Button2_Click 函数方法，返回
}
```

(3) 按钮“返回”的 Click 事件。双击【返回】按钮，添加代码，如下所示：

```
protected void Button2_Click(object sender, EventArgs e)
{
    if (Session["admin"] != null)
        Response.Redirect("TopicAdmin.aspx"); //如果是管理员则定向到 TopicAdmin.aspx
    else
        Response.Redirect("TopicList.aspx"); //如果是管理员则定向到 TopicList.aspx
}
```

(4) 添加用户是否登录代码。代码如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    Title = "发表新帖";
    if (!CheckUser()) //检查用户是否登录
        Response.Redirect("Login.aspx"); //如果没有登录则重新登录
}

private bool CheckUser() //此方法检查用户是否登录
{
    if (Session["login_name"] == null)
    {
        Response.Write("<Script>alert('请登录！');</Script>");
        return false;
    }
    return true;
}
```

P.9.3 查看详细信息页面设计

设计后的查看详细信息页面如图 P.14 所示。单击【回复】按钮回复帖，单击【返回】按钮返回到过去的页面。

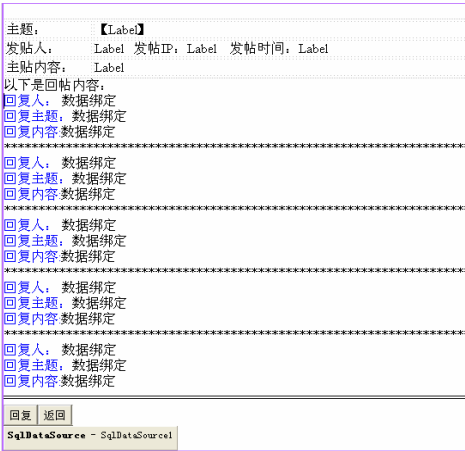


图 P.14 TopicDetail.aspx 设计后的部分截图

## 1. Repeater控件介绍

此页面涉及一个 Repeater 控件。Repeater 是一个可重复操作的控件，也就是说，它通过使用模板来显示一个数据源的内容，可以很容易地配置这些模板。Repeater 包含如标题和页脚这样的数据，它可以遍历所有的数据选项并应用到模板中。


Repeater 控件并不是由 WebControl 类派生而来的。所以，它不包括一些通用的格式属性，如控制字体、颜色等。然而，使用 Repeater 控件时，HTML（或者一个样式表）或者 ASP.NET 类可以处理这些属性。

Repeater 控件只支持模板。有以下的模板可供选择。

- (1) AlternatingItemTemplate: 指定如何显示其他每一选项。
- (2) ItemTemplate: 指定如何显示选项（AlternatingItemTemplate 可以覆盖这一模板）。
- (3) HeaderTemplate: 建立如何显示标题。
- (4) FooterTemplate: 建立如何显示页脚。
- (5) SeparatorTemplate: 指定如何显示不同选项之间的分隔符。

## 2. 设计步骤

(1) 设计页面。打开 TopicDetail.aspx 页面的设计视图，插入 3 行 2 列的表格。在表格的适当位置放入 5 个 Label 控件，分别用于显示主题、发帖人、发帖 IP、发帖时间及主帖内容。按图 P.12 所示添加提示字样。在表格下面从工具箱的数据栏中拖放 1 个 Repeater 控件，在 Repeater 控件下放入 2 个 Button 控件和 1 个 SqlDataSource 控件。Button1 和 Button2 的 Text 属性分别设置为“回复”和“返回”。

(2) 配置数据源。单击 SqlDataSource 控件的“”图标，选择“配置数据源”→在“配置数据源”对话框中选择“MyBBConnectionString”→单击【下一步】按钮→在“配置 Select 语句”对话框中选择“Reply”表，选择“Title”、“Content”、“CreateTime”和“UserLoginName”列→单击【where】→在“添加 WHERE 子句”对话框中设置 where 子句，如图 P.15 所示。单击【添加】按钮→单击【确定】按钮。

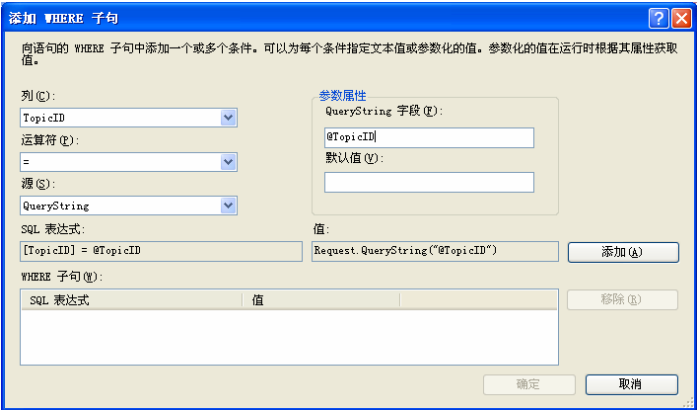


图 P.15 添加 WHERE 子句

(3) Repeater 控件的设置。切换到源视图，添加下面代码：

```
<asp:Repeater ID="Repeater1" runat="server" DataSourceID="SqlDataSource1">
```

```

<ItemTemplate>
    <asp:Label ID="Label6" runat="server" Text="回复人: " ForeColor="Blue"></asp:Label>
    <%# DataBinder.Eval(Container.DataItem, "UserLoginName")%><br />
    <asp:Label ID="Label7" runat="server" Text="回复主题: " ForeColor="Blue"></asp:Label>
    <%# DataBinder.Eval(Container.DataItem, "Title")%> <br />
    <asp:Label ID="Label8" runat="server" Text="回复内容:" ForeColor="Blue"></asp:Label>
    <%# DataBinder.Eval(Container.DataItem, "Content")%> <br />
</ItemTemplate>
<SeparatorTemplate>*****<br />
</SeparatorTemplate>
<FooterTemplate>=====</FooterTemplate>
</asp:Repeater>

```

(4) 按钮“回复”的 Click 事件。双击【回复】按钮，添加代码，首先添加命名空间“using System.Data.SqlClient;”，代码如下所示：

```

protected void Button1_Click(object sender, EventArgs e) //回复
{
    if (CheckUser()) //调用 CheckUser 方法确定是否登录
        Response.Redirect("TopicReply.aspx?topic_id=" Request.QueryString["topic_id"].ToString());
}

private bool CheckUser()
{
    if (Session["login_name"].ToString() == "guest") //如果是游客则不给回复
    {
        Response.Write("<Script Language=JavaScript>alert('对不起，你无此权限！');</Script>");
        return false;
    }
    return true;
}

```

(5) 按钮“返回”的 Click 事件。双击【返回】按钮，添加代码，表示返回到前面的页面，代码如下所示：

```

protected void Button2_Click(object sender, EventArgs e)
{
    if (Session["admin"] == null)
        Response.Redirect("TopicList.aspx");
    else
        Response.Redirect("TopicAdmin.aspx");
}

```

(6) 显示帖子的主题、发帖人等信息。当加载此页面时，从数据库中查询出主题、发帖人等信息，显示在页面上。代码如下所示：

```

protected void Page_Load(object sender, EventArgs e)
{
    if (!this.IsPostBack) //如果是第一次加载页面则执行下面代码

```

```

{
    int topicID = Convert.ToInt32(Request.QueryString["topic_id"]); //获取其他页面传来的 topic_id
    string constr = ConfigurationManager.ConnectionStrings["MyBBSCConnectionString"].ToString();
    string SqlStr = "Select UserLoginName, Title, IP, Content, CreateTime from Topic
        where TopicID=" + topicID + "";
    SqlConnection conn = new SqlConnection(constr);
    // 检查用户是否已存在
    SqlCommand Cmd = new SqlCommand(SqlStr, conn);
    conn.Open(); //打开数据库连接
    SqlDataReader sqlread = Cmd.ExecuteReader();
    if (sqlread.Read())
    {
        Label1.Text = sqlread["Title"].ToString();
        Label2.Text = sqlread["UserLoginName"].ToString();
        Label3.Text = sqlread["IP"].ToString();
        Label4.Text = sqlread["CreateTime"].ToString();
        Label5.Text = sqlread["Content"].ToString();
    }
    conn.Close(); //关闭数据库连接
}
}

```

## P.10 回帖模块设计

设计后的回复帖页面（TopicList.aspx）如图 P.16 所示。单击【回复】按钮回复帖，单击【返回】按钮返回到过去的页面。

图 P.16 设计后的 TopicList.aspx 的部分截图

设计步骤如下所示：

（1）设计页面。打开 TopicList.aspx 页面的设计视图，插入 4 行 2 列的表格。在表格的适当位置放入 1 个 Label 控件、2 个 TextBox 控件和 2 个 Button 控件，并输入提示字样。Label 控件的 Text 设置为“回复”，SkinID 设置为“skinLabel2”。TextBox1 的 SkinID 设置为“skinTextBox1”。TextBox2 的 SkinID 设置为“skinTextBox1”，TextMode 设置为“MultiLine”，见图 P.15。

（2）按钮“确定”的 Click 事件。双击【确定】按钮，添加代码，首先添加命名空间“using System.Data.SqlClient;”，代码如下所示：



```
protected void Button1_Click(object sender, EventArgs e)
{
    int topicID = Convert.ToInt32(Request.QueryString["topic_id"]); //获取其他页传来的 topic_id 值
    string constr = ConfigurationManager.ConnectionStrings["MyBBSConnectionString"].ToString();
    string SqlStr = "Insert into [Reply]([TopicID],[UserLoginName],[Title],[Content],[CreateTime],[IP])
        values(@TopicID,@UserLoginName,@Title,@Content,@CreateTime,@IP)";
    SqlConnection conn = new SqlConnection(constr);
    // 检查用户是否存在
    SqlCommand Cmd = new SqlCommand(SqlStr, conn);
    conn.Open(); //打开数据库连接
    // 添加参数对象,并给参数赋值
    Cmd.Parameters.Add("@TopicID", SqlDbType.Int).Value = topicID;
    Cmd.Parameters.Add("@UserLoginName", SqlDbType.VarChar, 50).Value = Session["login_name"];
    Cmd.Parameters.Add("@Title", SqlDbType.VarChar, 50).Value = TextBox1.Text;
    Cmd.Parameters.Add("@Content", SqlDbType.Text).Value = TextBox2.Text;
    Cmd.Parameters.Add("@CreateTime", SqlDbType.DateTime).Value = DateTime.Now;
    Cmd.Parameters.Add("@IP", SqlDbType.Char, 15).Value = Request.UserHostAddress.ToString();
    try
    {
        Cmd.ExecuteNonQuery(); //添加记录
    }
    catch (SqlException sqlException) //显示连接异常信息
    {
        Response.Write("<script>alert('" + sqlException.Message + "')</script>");
        return;
    }
    finally
    {
        conn.Close(); //关闭数据库连接
    }
    Button2_Click(null, null); //调用 Button2_Click 函数方法, 返回
}
}
```

(3) 按钮“返回”的 Click 事件。双击【返回】按钮，添加代码，如下所示：

```
protected void Button2_Click(object sender, EventArgs e)
{
    if (Session["admin"] != null)
        Response.Redirect("TopicAdmin.aspx"); //如果是管理员则定向到 TopicAdmin.aspx
    else
        Response.Redirect("TopicList.aspx"); //如果是管理员则定向到 TopicList.aspx
}
}
```

(4) 添加用户是否登录代码。代码如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!CheckUser()) //检查用户是否登录
        Response.Redirect("Login.aspx"); //如果没有登录则重新登录
}

private bool CheckUser() //此方法检查用户是否登录
{
    }
```

```

if (Session["login_name"] == null)
{
    Response.Write("<Script>alert('请登录! ');</Script>");
    return false;
}
return true;
}

```

## P.11 管理帖模块设计

设计后的管理帖页面（TopicAdmin.aspx）如图 P.17 所示，用于管理员管理帖，如果发现不良的帖则直接删除。

编号	标题	用户	发表时间	详细内容	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除
数据绑定	数据绑定	数据绑定	数据绑定	详细信息	删除

1 2

SqlDataSource1 - SqlDataSource1

Label1

图 P.17 TopicAdmin.aspx 设计后的部分截图

设计步骤如下所示：

- （1）设计页面。打开 TopicAdmin.aspx 页面的设计视图，从工具箱中拖放 1 个 GridView 控件和 1 个 SqlDataSource 控件，在 GridView 控件下再放入 1 个 ID 设置为“Label1”的 Label 控件。
- （2）配置数据源。单击 SqlDataSource 控件的“”图标，选择“配置数据源”→在“配置数据源”对话框中选择“MyBBSCConnectionString”→单击【下一步】按钮→在“配置 Select 语句”对话框中选择“Topic”表，选择“TopicID”、“UserLoginName”、“Title”和“CreateTiem”列→单击【下一步】按钮→单击【完成】按钮。
- （3）GridView 控件的设置。单击 GridView 控件的“”图标，在 GridView 任务中选择数据源为“SqlDataSource1”，选择“启用分页”和“启用排序”复选框。单击“编辑列”项，“HeaderText”分别设置为字段对应的汉字。添加 1 个 HyperLinkField，所对应的“HeaderText”设置为“详细内容”，“Text”设置为“详细内容”，“DataNavigateUrlFields”设置为“TopicID”，“DataNavigateUrlFormatString”设置为“TopicDetail.aspx?topic\_id={0}”，表示链接到 TopicDetail.aspx 页并把 TopicId 值传过去。添加 1 个 ButtonField，“HeaderText”和“Text”都设置为“删除”，“CommandName”设置为“D”，如图 P.18 所示。

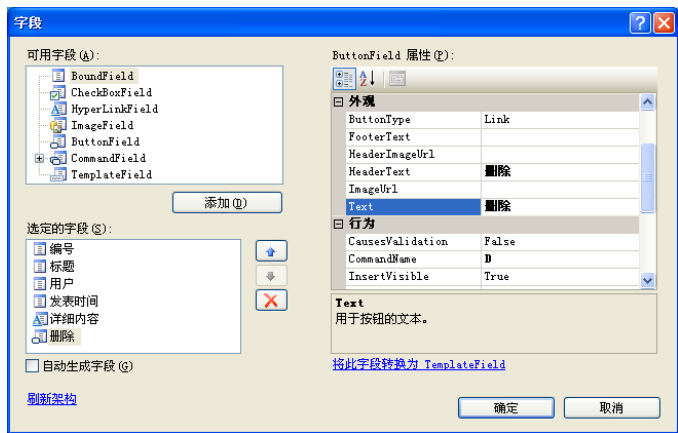



图 P.18 编辑 GridView 中的字段


(4) GridView 控件的 PageIndexChanging 事件。打开 GridView 控件的属性窗口，选择“”图标，双击“PageIndexChanging”空白处，添加代码，代码如下所示：

```
protected void GridView1_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    GridView1.PageIndex = e.NewPageIndex;
    GridView1.DataBind(); //重新绑定
    Label1.Text = "查询结果（第" + (GridView1.PageIndex + 1).ToString() + "页 共" +
        (GridView1.PageCount + 1).ToString() + "页）"; LabelPages.Text = "查询结果（第" +
        (GridView1.PageIndex + 1).ToString() + "页 共" + (GridView1.PageCount).ToString() + "页）";
}
```

(5) 添加显示所在页数代码。在 Page\_Load 方法体内添加代码，代码如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!this.CheckUser())
    {
        Response.Redirect("adminLogin.aspx"); //管理员未登录则重定向到 adminLogin.aspx
        GridView1.DataBind(); //重新绑定
        Label1.Text = "查询结果（第" + (GridView1.PageIndex + 1).ToString() + "页 共" +
            (GridView1.PageCount + 1).ToString() + "页）"; LabelPages.Text = "查询结果（第" +
            (GridView1.PageIndex + 1).ToString() + "页 共" + (GridView1.PageCount).ToString() + "页）";
    }

    private bool CheckUser() //此方法检测管理员是否登录
    {
        if (Session["admin"] == null)
        {
            Response.Write("<Script>alert('请登录! ');</Script>");
            return false;
        }
        return true;
    }
}
```

(6) 删除帖。打开 GridView 控件的属性窗口, 选择 “” 图标, 双击 “RowCommand” 空白处, 添加代码, 代码如下所示:

```
protected void GridView1_RowCommand(object sender, GridViewCommandEventArgs e)
{
    int index = Convert.ToInt32(e.CommandArgument); //待处理的行下标
    int topicId = -1;
    if (e.CommandName == "D") //如果命令是 “D” 则删除
    {
        topicId = Convert.ToInt32(GridView1.Rows[index].Cells[0].Text); //获取 TopicID
        GridView1_Delete(topicId); //调用 GridView1_Delete 方法删帖
    }
}

protected void GridView1_Delete(int topicId) //根据 TopicID 删除帖
{
    string sqlcon = ConfigurationManager.ConnectionStrings["MyBBSCConnectionString"].ToString();
    string sqlcomtext = "delete from [Topic] where TopicID='" + topicId + "'";
    SqlConnection conn = new SqlConnection(sqlcon);
    SqlCommand sqlcom = new SqlCommand(sqlcomtext, conn);
    try
    {
        conn.Open(); //打开数据库连接
        sqlcom.ExecuteNonQuery(); //执行
        GridView1.DataBind(); //GridView1 重新绑定
    }
    catch
    {
        Response.Write("<script>alert('出错')</script>");
    }
    finally
    {
        conn.Close(); //关闭数据库连接
    }
}
```

## P.12 系统扩展

本实例展示了小型 BBS 论坛系统的设计, 包含的功能较少, 离真正的论坛系统还有距离。在此基础上, 读者至少可以在以下几个方面做进一步的改进:

- (1) 增加搜索功能;
- (2) 实现分类论坛;
- (3) 增加用户权限设置 (如分类论坛版主设置);
- (4) 增加删除单个回复功能;
- (5) 增加帖子置顶功能;

上面列出的改进功能实现起来并不困难, 有兴趣的读者可以自己进行尝试。

# 反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为，歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：(010) 88254396; (010) 88258888

传 真：(010) 88254397

E-mail: dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

电子工业出版社总编办公室

邮 编：100036

# 《ASP.NET 3.5 教程》读者意见反馈表

尊敬的读者：

感谢您购买本书。为了能为您提供更优秀的教材，请您抽出宝贵的时间，将您的意见以下表的方式（可从 <http://www.huaxin.edu.cn> 下载本调查表）及时告知我们，以改进我们的服务。对采用您的意见进行修订的教材，我们将在该书的前言中进行说明并赠送您样书。

姓名：	_____	电话：	_____
职业：	_____	E-mail：	_____
邮编：	_____	通信地址：	_____

1. 您对本书的总体看法是：  

☐很满意    ☐比较满意    ☐尚可    ☐不太满意    ☐不满意
2. 您对本书的结构（章节）：

☐满意    ☐不满意    改进意见\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_
3. 您对本书的例题：

☐满意    ☐不满意    改进意见\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_
4. 您对本书的习题：

☐满意    ☐不满意    改进意见\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_
5. 您对本书的实训：

☐满意    ☐不满意    改进意见\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_
6. 您对本书其他的改进意见：

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_
7. 您感兴趣或希望增加的教材选题是：

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

请寄：100036    北京市万寿路 173 信箱高等教育分社    收  
电话：010-88254565            E-mail: gaozhi@phei.com.cn